UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES DE MADRID

TRABAJO FIN DE GRADO:

SISTEMA DE APRENDIZAJE EVOLUTIVO PARA UN ROBOT CAMINANTE

AUTOR: ÁNGEL LUIS JIMÉNEZ GARCÍA

TUTOR: ANTONIO BARRIENTOS CRUZ

Junio de 2015

DEPARTAMENTO DE AUTOMÁTICA, INGENIERÍA ELECTRÓNICA E INFORMÁTICA INDUSTRIAL
División de Ingeniería de Sistemas y Automática

“Los imposibles de hoy serán posibles mañana.”

Konstantin Tsiolkovsky

Agradecimientos

A *Antonio*, mi tutor, por brindarme la oportunidad de trabajar en el campo de los algoritmos evolutivos y abrirme las puertas a un mundo nuevo. Por su ayuda y orientación cuando más lo he necesitado. Sin duda el mejor profesor que he tenido en todos estos años.

A mi *familia*, a mis tíos, primos y abuelos por su constante interés y por el gran apoyo cada vez que me he visto falto de fuerzas. Especial agradecimiento a *mis padres*, ya que sin ellos nada de esto habría sido posible. Gracias por vuestra impagable ayuda. Por apoyarme, orientarme y aguantarme día a día y gracias por vuestro enorme esfuerzo en todos los sentidos para que pueda estar donde estoy hoy. Gracias por ser los mejores padres que uno puede desear.

A *Lorena*, por toda la paciencia que has demostrado conmigo todos estos años. Por tus ánimos, tu apoyo incondicional y tu confianza en mí. Por todas esas lágrimas derramadas que supiste calmar y todas esas sonrisas que me supiste sacar mostrándome así el lado alegre de la vida. Por estar ahí cada día en lo bueno y en lo malo, sin ti no hubiera llegado a la consecución de todo esto. En definitiva, gracias por ser una pieza clave en mi vida.

A *Luis*, por haber sido una persona muy importante de cara al logro de este objetivo. Gracias por tu inestimable ayuda, consejos, apuntes, compañía y todas aquellas cosas que hacen de ti un verdadero compañero de aventuras. También a *David* por haber estado ahí siempre para lo que he necesitado, por tu compañía en tantas clases y por todos esos buenos momentos. Gracias a ambos por vuestra ayuda para que este documento luzca tan bonito.

A *Pablo, Samuel, Miguel* y *Roberto* por todos esos largos días en la universidad y también por aquellos fuera de ella que tan buenos recuerdos me dejan.

A *Nacho, Óscar* y *Jonás* por su preciada amistad y todos sus valiosos consejos y ánimos. Gracias por confiar en mí y saber siempre como hacer que me levante después de haber caído. Muchas cosas hemos vivido juntos y espero que podamos vivir muchas más.

A mis amigos de Coslada: *Adrián, Arturo, D.Baeza, Carmen, Cristina, Jesús, Lidia, C.Múgica, D.Elvira, Roberto, Álvaro, Laura, D.Mulero, Nuria, Sara, Silvia* y *C.Gras* por todo vuestro apoyo, vuestros ánimos cuando lo necesitaba y por vuestra extrema sinceridad. En especial gracias a *D.Gallo* por toda la documentación y ayuda proporcionada al comienzo de este proyecto y gracias a *Marta* por acompañarme en este tren desde el principio y compartir sufrimiento.

A *Fernando*, que en este último año he descubierto la gran persona y lo buen amigo que eres. Amigos así no se encuentran a menudo y espero seguir contando con tu amistad muchos años más.

A *Susana, Marta, Jorge, Juan, Laura, Beatriz, Carmina* y todos esos amigos de la universidad que me han acompañado desde el principio y se han ganado estar entre estas líneas.

A *J.Misas*, por su buen hacer y su ayuda con cualquier problema administrativo que me surgía. Por su atención, simpatía y profesionalidad.

A todos aquellos que me dejo por el camino, esperando que no sean muchos.

A todos y cada uno de vosotros, muchas gracias.

Ángel Luis Jiménez García
Madrid, Junio de 2015

Resumen del Trabajo

Alcance del proyecto

Este proyecto consiste en el desarrollo de un sistema de aprendizaje evolutivo para un robot caminante, más concretamente un robot caminante cuadrúpedo con dos grados de libertad en cada pata. Para ello, se utiliza Matlab en sintonía con un software de simulación robótica llamado V-REP (Virtual Robot Experimentation Platform). El robot debe aprender a caminar por sí mismo con secuencias de marcha que produzcan un desplazamiento efectivo de su centro de masa, sin que él tenga conocimiento alguno de su cinemática, ni haber sido programado específicamente para caminar de una determinada manera, haciendo uso para ello de algoritmos evolutivos basados en genética.

Introducción

El aprendizaje robótico es importante si no se necesita un robot para una tarea específica, si no que se tiene un entorno que se encuentra en contínuo cambio. Muchas aplicaciones requieren el uso de robots capaces de moverse y desplazarse para la consecución de sus objetivos, teniéndolo que hacer muchas veces por terrenos irregulares y con obstáculos. El aprendizaje de robots es, por tanto, una manera de adaptarse a distintas situaciones y los algoritmos evolutivos son una técnica a utilizar para que los robots puedan aprender por sí mismos.

La esencia de los algoritmos evolutivos consiste en tomar como ejemplo la genética inherente en la evolución de los seres vivos para solucionar complejos problemas de optimización, para cualquier campo de aplicación. En dicha evolución animal el individuo que mejor se adapta al entorno y a las necesidades es el que sobrevive. En el caso de los algoritmos evolutivos se tiene un conjunto de individuos, también conocido como población, inicialmente aleatoria, donde cada uno de los individuos representa una posible solución al problema a resolver. Basándose en los esquemas propuestos por Darwin sobre la selección natural, los individuos se irán adaptando mejor a la solución requerida con el paso de las generaciones.

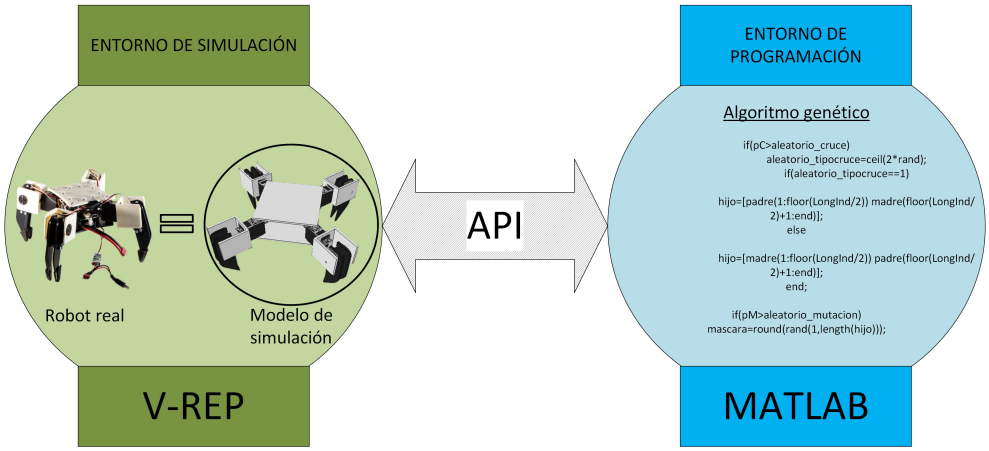
La realización de experimentos de un algoritmo genético requiere gran cantidad de tiempo, debido a que se debe probar, en cada generación, individuo por individuo en el robot y obtener la bondad de cada uno, también llamado *fitness*, para un correcto funcionamiento. En consecuencia, no es viable probar cada combinación en el robot real, siendo más eficiente la realización de simulaciones para acelerar así el proceso. Una vez obtenidos los resultados de las simulaciones, se pasa a experimentar con un robot real y a comprobar las distintas soluciones. La realización de las simulaciones se efectúan en V-REP debido a la comodidad de su uso, su versatilidad y la posibilidad de comunicarse con Matlab.

La evolución de la naturaleza, y más concretamente la evolución del ser humano, hace totalmente necesario llevar a cabo proyectos en los cuales la evolución esté presente. Es destacable que el afán del ser humano de tomar ejemplo de la sabia naturaleza conduce a que el aprendizaje en el mundo de los robots tome un papel de suma importancia, al brindar la oportunidad de construir un mundo adaptado a nuestras necesidades.

Software

Los programas y métodos de comunicación que se utilizan para las simulaciones son los siguientes:

- Matlab: Es el programa donde se corre el algoritmo genético, el entorno de programación. Es, por tanto, el encargado de hacer todos los cálculos y recoger resultados. Matlab se encarga de seleccionar, cruzar y sustituir los individuos de las poblaciones de las distintas generaciones, mandando para ello órdenes al entorno de simulación.
- V-REP: En este entorno de simulación se encuentra el modelo del robot real desarrollado. El modelo recibe órdenes de movimiento desde Matlab, que son ejecutadas tras su recepción.
- API: Las comunicaciones entre Matlab y V-REP se realizan a través de una API disponible entre ambos programas. El entorno de simulación hace de servidor y el entorno de programación realiza la función de cliente, solicitando las comunicaciones.



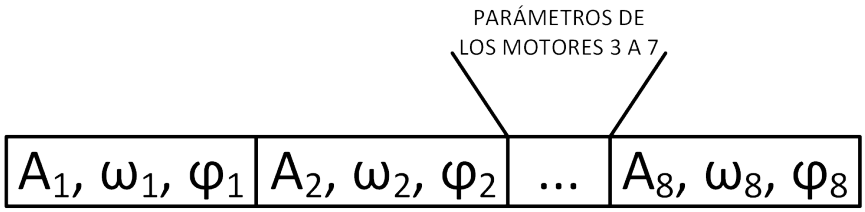
El procedimiento de simulación consiste en que, tras poner en marcha el servidor en el lado de V-REP, Matlab comienza a correr el algoritmo genético. En el momento en que se hace necesario mover el modelo de simulación para evaluar la bondad de un individuo (el avance que logra), Matlab solicita a V-REP el valor de la posición inicial y después manda las órdenes de movimiento pertinenes a través de la API.

Tras moverse el modelo de simulación, Matlab solicita a V-REP el valor de la posición final alcanzada, para continuar con el desarrollo del algoritmo. Este proceso se realiza con cada individuo en cada generación, pues es necesario evaluar la bondad de cada uno en relación a la función objetivo, que es el avance del robot.

Algoritmo Genético

Un algoritmo genético es una manera de solucionar problemas de optimización, que toma como ejemplo la evolución. Se compone de una población de individuos, que inicialmente son generados de manera aleatoria. El problema se caracteriza a través de parámetros que definen diferentes aspectos importantes de la solución. Cada individuo representa una solución distinta al problema y estará codificado de manera que los parámetros van uno tras otro formando de esta manera el cromosoma. El conjunto de parámetros que componen el cromosoma de cada individuo, en el presente trabajo, define una secuencia de marcha del robot. Cada individuo tendrá asociado un valor de *fitness*, que define cómo de bien soluciona ese individuo el problema. Los individuos se irán cruzando en cada generación y la población irá avanzando, en principio, hacia soluciones mejores. Además, podrán producirse mutaciones con una cierta probabilidad.

La codificación de los individuos es una cuestión importante, pues el cromosoma debe componer una secuencia de marcha del robot que lo hará avanzar. Tras considerar diferentes alternativas, se decide que el cromosoma que representa cada individuo estará formado por 24 parámetros, 3 parámetros para cada uno de los 8 motores del robot. Los tres parámetros de cada motor definen su movimiento, que seguirá una senoide, y son la amplitud A, la frecuencia ω y el retraso φ. En la siguiente figura se ilustra el aspecto del cromosoma de un individuo.



De esta manera, se tiene una codificación en números no binarios y con la que cada motor se moverá hacia delante y luego hacia atrás contínuamente, siguiendo un movimiento senoidal. En la mayoría de las publicaciones se trabaja con codificaciones binarias, por lo que se considera de interés el uso de una codificación no binaria que suponga una aportación relevante al uso más frecuente de los algoritmos genéticos.

La selección de los individuos se realiza reproduciendo la población hasta el doble y luego escogiendo la mejor mitad, de manera que la población tiene siempre un número constante de individuos. Para la reproducción se realizan cruces en un punto y con una baja probabilidad los hijos sufrirán mutaciones. A continuación se muestra un pseudocódigo del algoritmo genético principal desarrollado, donde los criterios de finalización son que la población converja o bien que se alcance el número límite de generaciones:

```
Generar poblacion inicial aleatoria

MIENTRAS no se cumpla el criterio de finalizacion

        MIENTRAS la poblacion no sea el doble
                cruzar dos progenitores con probabilidad Pc
                SI se ha producido el cruce
                        mutar descendiente con probabilidad Pm
                SINO
                        copiar un progenitor en el descendiente
                FIN SI
                incluir descendiente en la poblacion
        FIN MIENTRAS

        evaluar bondad de los individuos de la poblacion
        escoger la mejor mitad para la siguiente generacion

FIN MIENTRAS
```

Simulaciones

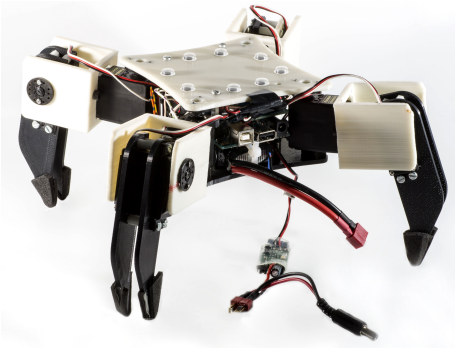
Existen modelos de simulación ya desarrollados que podrían utilizarse, pero se ha creído conveniente realizar un modelo a medida para dotar de mayor realidad y exactitud al trabajo. El modelo de simulación desarrollado se diseña pieza por pieza en Autocad, para posteriormente ensamblar todas las partes en V-REP. Con el objetivo de comprobar que el modelo es completamente funcional, se comprueba que todos los motores se mueven correctamente y que es posible programar una secuencia de marcha para que el robot la lleve a cabo. Tras las pruebas de movimiento, se realizan pruebas de comunicación entre V-REP y Matlab.

Las simulaciones se han realizado variando y combinando los parámetros de control del sistema, que son el número de generaciones, el número de individuos, la tasa de cruce y la tasa de mutación. Debido a la naturaleza aleatoria inicial de un algoritmo genético, de cada prueba se realizan varias repeticiones con el fin de trabajar con la media resultante.

Con los datos obtenidos de todas las simulaciones realizadas se han llevado a cabo ciertos análisis. En dichos análisis se pretende observar: la variación del *fitness* del individuo más apto en función de la variación de los parámetros de control, el mejor cruce, la mutación menos perjudicial y, evidentemente, la combinación que ofrezca mejores resultados.

Robot real

El robot del que se dispone fue construido en un proyecto anterior, tomando el relevo en el presente trabajo. Se trata de un robot caminante cuadrúpedo, con dos grados de libertad en cada pata. El control del mismo se lleva a cabo mediante el uso de un Arduino Mega ADK. El movimiento se consigue con 8 servos y la alimentación se realiza a través de baterías.



Se ha trasladado, con resultados satisfactorios, lo obtenido en las simulaciones y se ha probado en la realidad haciendo uso de este robot. En definitiva, el objetivo de estos experimentos es comprobar que se produce un aprendizaje en el robot que provoca que se desplace, en relación a los parámetros de control escogidos. En concreto se han realizado tres tipos de experimentos:

1. Peor solución: Se ha probado la peor solución obtenida en las simulaciones y se ha analizado por qué es la peor.
2. Solución intermedia: Se ha probado una solución a camino entre la peor y la mejor.
3. Mejores soluciones: De las simulaciones se obtienen dos soluciones con resultados altamente parecidos, que pasan a probarse en la realidad. Además, se analiza por qué estas soluciones arrojan mejores resultados.

Palabras clave

Algoritmo evolutivo, simulación, robot caminante, aprendizaje robótico, Matlab, V-REP.

Códigos UNESCO

- 1203 Ciencia de los ordenadores: 1203.02 Lenguajes Algorítmicos; 1203.04 Inteligencia Artificial; 1203.26 Simulación.
- 1207 Investigación Operativa: 1207.02 Sistemas de Control.
- 3311 Tecnología de la instrumentación: 3311.01 Tecnología de la Automatización; 3311.02 Ingeniería de Control.

Índice general

Lista de Figuras

Lista de Tablas

1

Introducción

1.1. Motivación

La realización de este Trabajo Fin de Grado surge en primera instancia del interés del autor por el creciente desarrollo tecnológico y la evolución en sistemas inteligentes. La asignatura de Robótica impartida en cuarto curso de la especialidad de Automática-Electrónica avivó aún más el deseo de investigar y trabajar en este campo. De esta motivación surge el contacto con el profesor, *Antonio Barrientos*, quien en una reunión ofrece diversos e interesantes trabajos para introducirse en la investigación sobre el desarrollo de aplicaciones relacionadas con, entre otros, la cooperación robótica, la genética evolutiva e incluso proyectos en colaboración con otros campos de estudio como la psicología o el deporte. Aunque son los algoritmos evolutivos, del amplio abanico ofertado, los que atraen especialmente al autor debido a la gran versatilidad que ofrecen.

En la mayoría de aplicaciones los robots son diseñados y programados para la realización de una o varias tareas concretas. Un claro ejemplo de ello son los brazos robóticos que se utilizan en algunas fábricas para automatizar y mejorar los procesos de producción. Muchas aplicaciones requieren el uso de robots capaces de moverse y desplazarse para la consecución de sus objetivos, siendo aquí donde toma importancia el **aprendizaje en robots**.

Cuando un robot debe desplazarse por zonas irregulares, puede resultar complejo programarlo para que sea capaz de moverse en distintos tipos de superficies y terrenos. Se haría necesario entonces, reprogramarlo cada vez que el robot cambie de terreno. Pero, ¿qué ocurriría si el robot fuese capaz de aprender a caminar autónomamente en distintos terrenos? En dicho caso, sería más sencillo desplazarse por superficies cambiantes y superar obstáculos. El aprendizaje de robots es, por tanto, una manera de adaptarse a distintas situaciones y los algoritmos evolutivos son una técnica que se puede utilizar para que los robots puedan aprender por sí mismos. Mediante dicho aprendizaje existe además, la ventaja de entender mejor el comportamiento inteligente existente en la naturaleza.

Por otra parte, también puede ser interesante el uso de algoritmos evolutivos para superar los daños sufridos en un robot. Si un robot caminante se ve dañado en una de sus patas, por ejemplo, podría investigar por sí mismo la mejor manera de desplazarse sin esa pata que no puede utilizar debido al daño sufrido. En este sentido hay ciertos investigadores trabajando en algunos proyectos sobre la resistencia de robots caminantes, como *Jean-Baptiste Mouret* [1] [2] [3] o *Hod Lipson* [4] [5].

Los **algoritmos evolutivos** han demostrado que para solucionar complejos problemas de optimización pueden ser herramientas poderosas, se trate del campo de aplicación que se trate. El campo de investigación de los algoritmos evolutivos se nutre y apoya en varias áreas de

conocimiento entre las que se encuentran, entre otras, la biología, la inteligencia artificial, la optimización, la programación y la toma de decisiones.

De todas ellas la que muestra una influencia más significativa es **la biología**, debido a que un algoritmo evolutivo es un método adaptativo que se basa en la reproducción sexual y en el principio de superviviencia del más apto [6][7]. En dicha evolución animal el individuo que sobrevive es el que mejor se adapta al entorno y a las necesidades. En el caso de los algoritmos evolutivos se tiene un conjunto (también conocido como población) de individuos, inicialmente aleatorios, donde cada uno representa una posible solución al problema a resolver. Basándose en los esquemas propuestos por *Darwin* sobre la selección natural, los individuos se irán adaptando mejor a la solución requerida con el paso de las generaciones [8].

Uno de los mayores retos que se presentan a la hora de trabajar con este tipo de algoritmos es la codificación del problema que se tiene entre manos. Dicha codificación no es trivial en la mayoría de los casos y debe buscar la superación de los problemas de carácter paramétrico, enfocando así la solución del problema de optimización de una manera creativa y eficaz. Para ello se debe pensar en los parámetros que tiene el problema y la manera más eficiente de organizar los datos que conforman el individuo solución. Otro gran reto comparable a la codificación, si no mayor, es la elección de la función de coste. La función de coste, también denominada función de fitness, es la función que determina cómo de adecuado es un individuo a la solución requerida. Los individuos se irán ordenando y reproduciendo en las distintas generaciones en función de su valor de fitness y por tanto la elección de una buena función de coste es crucial.

Existen más factores a tener en cuenta en adición a los dos mentados anteriormente como son el tamaño de la población o las tasas de cruce y de mutación, así como el tipo de mutación a realizar en los infrecuentes casos en que aquélla se deba producir.

La conjunción de la genética evolutiva con otros métodos de resolución de problemas es habitual. Cuando el problema que se quiere resolver no admite un tratamiento algorítmico se hace uso de otros enfoques, como por ejemplo las redes neuronales (ver Figura 1.1).

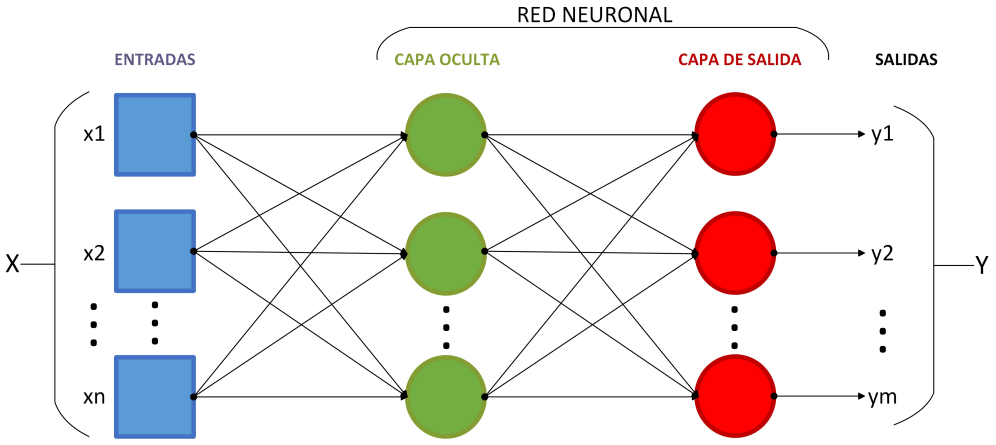


Figura 1.1: *Ejemplo de Red Neuronal*

Las **redes neuronales**, son importantes a tener en cuenta como otro modo de ver el problema. Una red neuronal trata de reproducir la forma en que el cerebro humano utiliza la experiencia pasada para la resolución de una situación. En la redes neuronales se presentan varias posibilidades neuropsicológicas en el sentido de que para cada red que se comporta de una manera, existe otra red que se comporta de manera distinta pero llegando a los mismos resultados [9].

Con todas sus ventajas y desventajas, los evolutivos son unos de los algoritmos de optimización más ampliamente utilizados en la moderna optimización no lineal [10]. De todo esto el autor obtiene una profunda motivación y un creciente ánimo hacia la investigación, aprendizaje y desarrollo del mundo de los *Algoritmos Evolutivos*.

1.2. Objetivos del Trabajo Fin de Grado

El objetivo del presente proyecto queda reflejado perfectamente en el título del mismo y es **el desarrollo de un sistema de aprendizaje evolutivo para un robot caminante**. Lo que se pretende es el desarrollo y la evaluación de algoritmos evolutivos para que un robot, sin previo conocimiento alguno de su cinemática, encuentre secuencias de marcha que produzcan un desplazamiento efectivo sobre su centro de masa. Tras la realización de un elevado número de simulaciones, con los parámetros debidamente ajustados, se deberán evaluar los resultados sobre un robot real. Una vez conocido el objetivo principal, se puede descomponer el mismo en distintos objetivos más concretos.

Para la consecución del objetivo, a parte de una profunda investigación del estado del arte, es necesario realizar simulaciones. Por tanto, se deberá **realizar un modelo y un entorno de simulación amigables**. Dicho modelo de simulación deberá ser, además, lo más fiel posible al robot real. En cuanto a la parte evolutiva del proyecto, se deberá **codificar un algoritmo genético** que consiga, mediante la evolución de soluciones posibles, hacer avanzar al robot caminante en la simulación. Por último, se deberán **trasladar los resultados de las simulaciones al robot real** para contrastar las soluciones obtenidas.

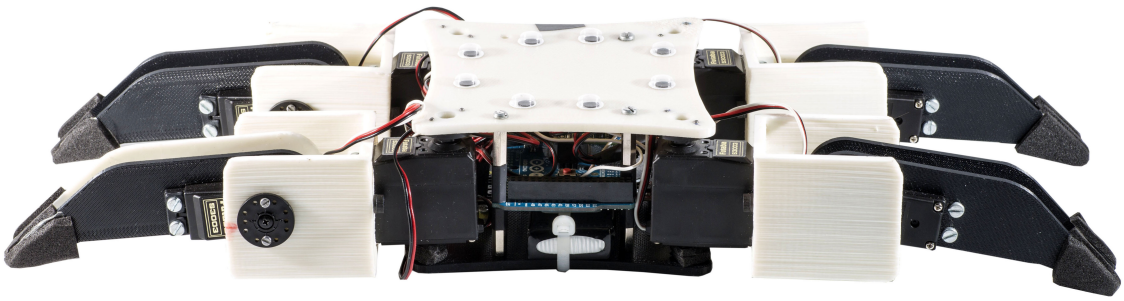


Figura 1.2: *Perfil del robot tumbado*

El robot del que se dispone se muestra en la Figura 1.2 y fue construido en un trabajo anterior [11]. Es un robot cuadrúpedo en el cual cada pata tiene dos grados de libertad. En la Figuras 1.3 y 1.4 se muestran más imágenes del robot.

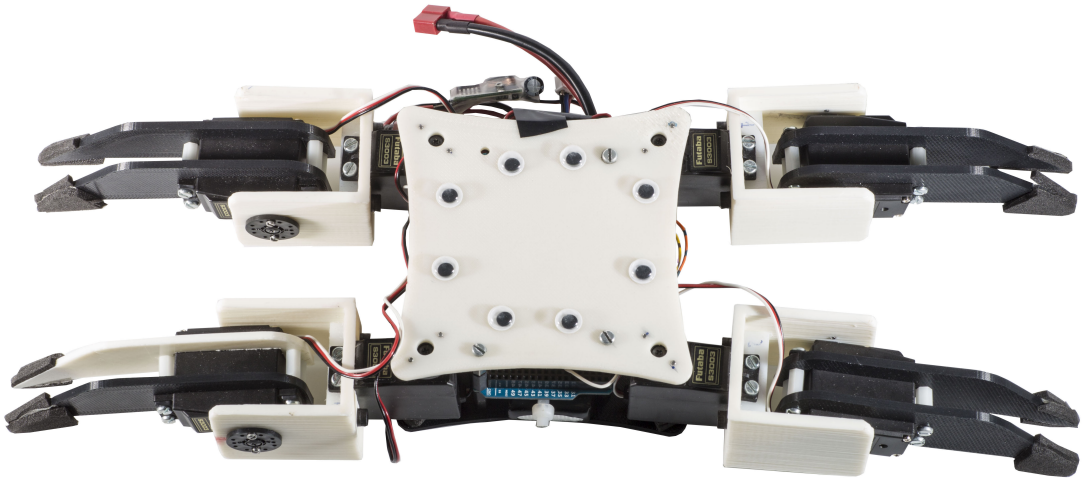


Figura 1.3: *Planta del robot tumbado*

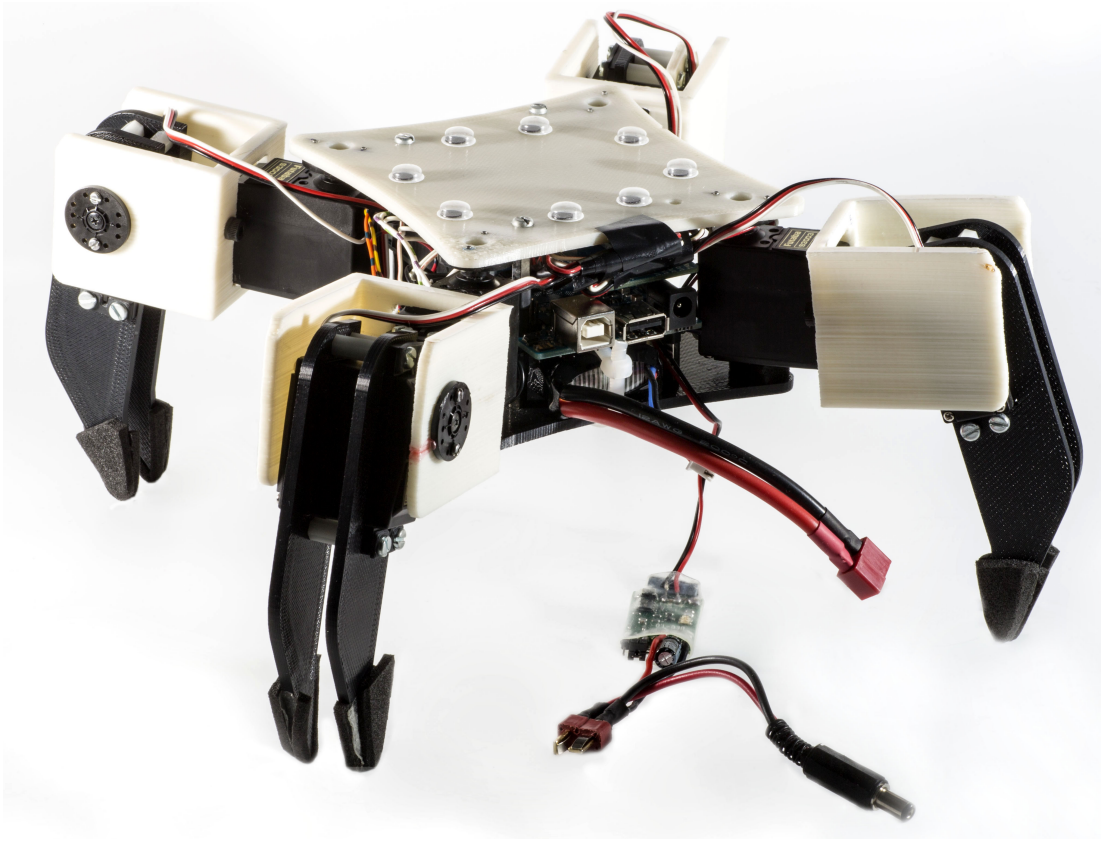


Figura 1.4: *Robot levantado*

2

Estado del arte

2.1. Orígenes de la computación evolutiva

Durante años se ha observado a los animales. Se ha comprobado que las distintas especies animales durante el paso de distintas generaciones se van adaptando al medio en el que habitan, que suele ser un medio que cambia de manera frecuente, con el objetivo de sobrevivir del mejor modo posible. Un naturalista inglés llamado *Charles Robert Darwin* es considerado como el mayor investigador en el campo de la evolución animal y la selección natural. Según *Darwin* el miembro con mejores aptitudes útiles de cara a la supervivencia era el que tendría unas mayores posibilidades de reproducirse y, por tanto, de conservar sus genes. Estos genes, a su vez, mejorarían con la generación posterior y así de manera sucesiva hasta llegar a un individuo altamente adaptado a las necesidades de supervivencia [12].

Del mismo modo que las diferentes especies se adaptan, podría pensarse en utilizar las teorías sobre la selección natural a la optimización de problemas complejos que de otro modo sería tedioso abordar. En este caso se podría tener una población de distintos individuos donde cada uno representara una posible solución al problema que se está tratando, siendo cada solución distinta como primera asunción. De cada individuo se irían escogiendo los mejores de ellos con el paso de las generaciones, tomando como referencia la adaptación al problema que se quiera solucionar [13][14].

En la Figura 2.1 se tiene una clasificación de lo que se conoce como **Soft Computing**, término recientemente acuñado que recoge el uso emergente de muchas disciplinas de la computación. Entre los campos de investigación implicados se encuentra la computación evolutiva, la cual se desarrollará a continuación.

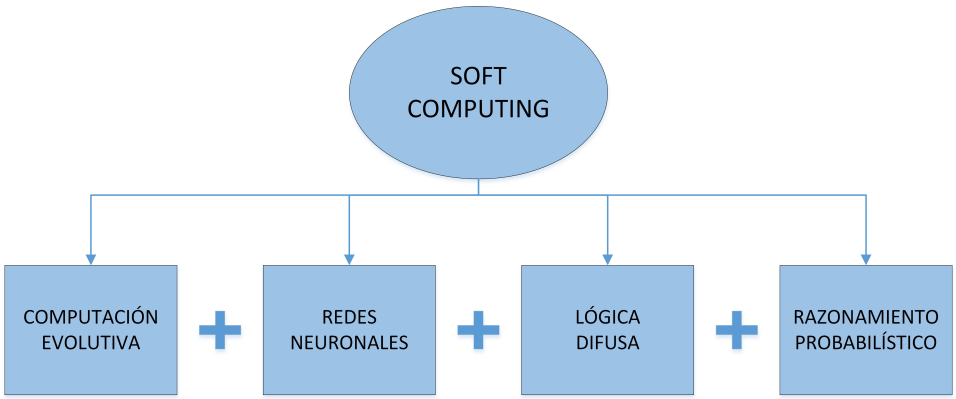


Figura 2.1: *Soft Computing*

La **Computación Evolutiva** se podría definir, de una manera genérica, como un conjunto de modelos computacionales que se inspiran en la evolución y en la selección natural. Profundizando un poco más en la descripción, el término de computación evolutiva hace referencia al estudio basado en la evolución de los fundamentos y aplicaciones de determinados sistemas heurísticos [15]. Se pueden clasificar dichos sistemas heurísticos de manera que juntos componen la llamada computación evolutiva.

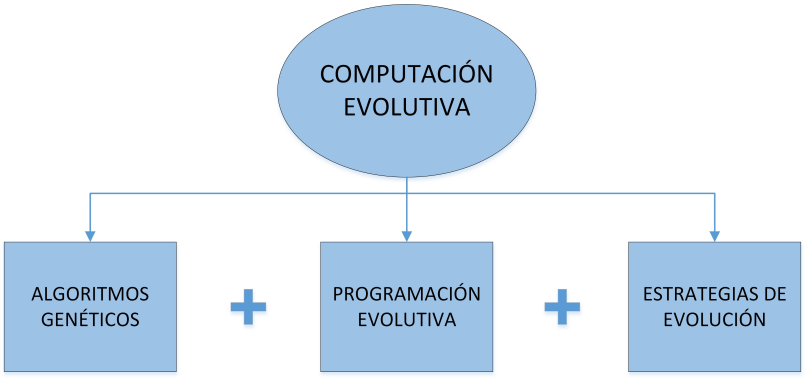


Figura 2.2: *Clasificación de los sistemas heurísticos en computación evolutiva*

Un **Algoritmo Genético** consiste en una población de posibles soluciones a un problema codificadas en cromosomas. A cada uno de esos cromosomas se le asignará un valor de bondad, ajuste o fitness que representará su efectividad como solución al problema. En función de este ajuste cada cromosoma tendrá más o menos probabilidades de reproducirse. En todo este proceso también estarán presentes las mutaciones, que son modificaciones en los genes que se realizarán con una probabilidad que en la mayoría de los casos es pequeña.

Un investigador de la Universidad de Michigan llamado *John Holland* fue de los primeros en lanzarse a combinar la selección natural con la computación, siendo así una figura importante para el desarrollo de los algoritmos genéticos. A finales de los años 60 desarrolló una técnica que permitió incorporar la selección natural a un programa. Su objetivo era lograr que las computadoras que usaran su técnica aprendieran por sí mismas. Originalmente se bautizó dicha técnica como **planes reproductivos**. No obstante después de que en 1975 *Holland* publicase su libro [16], comenzó a ganar popularidad el nombre de **Algoritmos Genéticos**.

Tras el trabajo realizado por *Holland*, diversos investigadores tomaron sus técnicas para implementarlas en programas evolutivos. Entre otros, *Smith* lo aplicó a problemas de optimización heurísticos [17], *Cory Fujiki* al dilema del prisionero [18] y *Joseph Hicklin* a la evolución de expresiones LISP para programas capaces de resolver juegos sencillos [19].

La **Programación Evolutiva** empezó a tomar fuerza como consecuencia del trabajo publicado por *Lawrence J. Fogel*, *Alvin J. Owens* y *Michael John Walsh* en 1966 titulado *«Artificial Intelligence Through Simulated Evolution»*[20]. En dicho trabajo se observa que el tratamiento

de la Programación Evolutiva es similar al de los Algoritmos Genéticos, cambiando la representación de los individuos. Aquí, los individuos son denominados organismos y son máquinas de estado finito. Son representados como una terna formada por:

- Valor de estado actual.
- Símbolo del alfabeto utilizado.
- Valor del nuevo estado.

El funcionamiento de estas máquinas de estado finito es simple: Encontrándose en el estado actual, se toma el valor del símbolo actual y si coincide con el símbolo de nuestra terna nos movemos al nuevo estado. Antes de producirse los cruces se relizan mutaciones sobre los organismos padres de la futura descendencia, después se evalúa toda la población y solo se reproducirán aquellos organismos que resuelvan mejor alguna de las funciones objetivo marcadas.

Las **Estrategias de Evolución** son unas técnicas de optimización creadas en torno a la década de los años 60. Más adelante tomaron gran importancia las obras de *Ingo Rechenberg* y *Hans-Paul Schwefel* [21][22]. En 1971 *Rechenberg* propone el modelo (μ+1)-ES, incorporando el concepto de población desarrollado con anterioridad. En este modelo μ padres generan únicamente un solo hijo, utilizando el operador de mutación, que reemplaza a su peor padre en la siguiente generación. En 1974 *Schwefel* retoma la obra de *Rechenberg* y propone dos modelos en los que se generan más de un hijo. Estos dos modelos son:

- Modelo (μ+λ)-ES: La selección se realiza entre todos, por lo que padres e hijos compiten y solo pasan a la siguiente generación los μ mejores individuos.
- Modelo (μ,λ)-ES: La selección se realiza únicamente entre los hijos y los padres son totalmente descartados. Los hijos compiten entre ellos y pasan a la siguiente generación los μ mejores.

A pesar de su elevada importancia, hasta los años 80 estos modelos no fueron investigados de manera más profunda. En esta década, resurgió el interés por estos modelos de estrategias de evolución debido al desarrollo de las computadoras paralelas. Varios esquemas que aplicaban los conceptos de paralelismo, optimización multiobjetivo y los esquemas de adaptación fueron desarrollados en los años 90.

2.2. Bases biológicas

La madre naturaleza es sabia y se adapta a los cambios y necesidades que se van produciendo. Basta observar a los seres vivos y las distintas especies animales para darse cuenta de ello. Su organización y evolución son de admirar. Todo gran invento surge de la observación y la posterior postulación de una o varias preguntas. En nuestro caso nos podríamos preguntar: ¿por qué no imitar el comportamiento de los seres vivos para solucionar problemas complejos? En efecto, eso es lo que hacen los Algoritmos Genéticos, inspirarse en la evolución y tomarla como base.

2.2.1. Estructura genética

Para lograr entender la teoría de la evolución hay que hablar sobre la información genética y la manera de transferirse dicha información. La unidad de transferencia en la herencia son los **genes**, los cuales se encuentran en el **genotipo** del individuo. Como puede observarse en la Figura 2.3, los genes están ordenados jerárquicamente en cada **cromosoma** y el genotipo consiste en un conjunto de cromosomas. Todo esto conforma el sistema de tranferencia de información en la herencia producida en la reproducción.

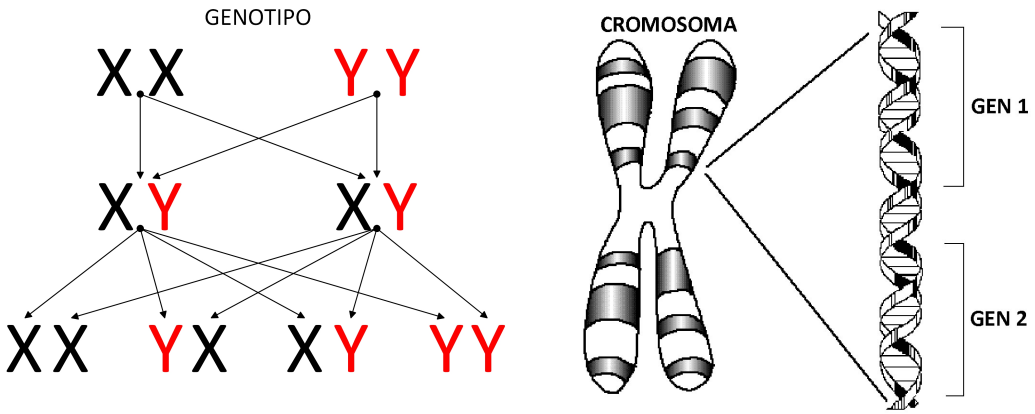


Figura 2.3: *Genotipo, cromosoma y genes*

Por tanto los genes son la unidad básica de información genética. La apariencia visible y la funcionalidad de un individuo aparece reflejada en la información contenida en el genotipo y se conoce como fenotipo. En estos fenotipos se producen ocasionalmente alteraciones que se conocen como mutaciones. Las mutaciones son variaciones de uno o más genes y tienen un carácter aleatorio. Si la mutación producida da muestras de tener unas buenas características permanecerá en las siguientes generaciones del proceso evolutivo, a través de la selección del individuo. Si la mutación producida no da muestras de tener buenas características desaparecerá

en próximas generaciones. Se hace necesario apuntar que las mutaciones no parecen tener un propósito claro.

2.2.2. Selección de individuos en la naturaleza

En la naturaleza se produce una constante competición por los recursos disponibles entre los distintos individuos de una población. Esta competición da lugar a la **selección natural**, motivada por la necesidad de producir descendencia. Los individuos que consigan mejores resultados en la obtención de recursos serán los que tengan mayores probabilidades de reproducirse y por tanto sus genes serán transferidos a un mayor número de descendientes. Por otro lado los individuos menos preparados tendrán un menor número de descendientes, pudiendo llegar a no tener ninguno. De esta manera los genes de los individuos más aptos se propagarán a un número cada vez mayor de descendientes en las sucesivas generaciones.

La existencia de abundantes recursos puede producir una evolución exponencial de la población. Esto implica una alta probabilidad de que los descendientes estén mejor adaptados que sus ancestros, puesto que obtienen una combinación de sus mejores genes. La presión selectiva suele resultar, por tanto, en una continua mejora en la que las especies evolucionan cada vez más adaptándose mejor al medio ambiente a medida que van transcurriendo las distintas generaciones [23].

Por supuesto la adaptación de un individuo al medio también está determinada por su **aprendizaje**, y no solamente por sus genes. Generalmente el aprendizaje se produce, o bien mediante prueba y error, o bien mediante imitación de la conducta de los progenitores. Este aprendizaje se puede implementar en los Algoritmos Genéticos con técnicas como el ajuste fino. El ajuste fino consiste en crear una población a partir del individuo al cual se le desea incorporar el aprendizaje, produciendo en éste ligeras modificaciones para generar la población. El individuo resultante del ajuste fino será básicamente el mismo individuo de origen, pero con el aprendizaje incorporado.

2.2.3. Características de los procesos evolutivos

Se han visto hasta ahora las características de la evolución natural. En el mundo de los algoritmos evolutivos las características no son del todo exactamente iguales a las de la naturaleza. Esto es debido a que se asemeja más a una evolución forzada por el hombre sobre otras especies animales. En estos casos se conocen la adaptación exigida y las estructuras de información. También están preestablecidas la función de aptitud y las operaciones a realizar [24].

En la Tabla 2.1 se recogen algunas características relevantes y sus diferencias entre la evolución natural y la computación evolutiva.

Evolución Natural	Computación Evolutiva
Autonomía operativa	Autonomía operativa
Procesos controlados y no controlados	Procesos controlados
Reproducción sexual y asexual	Múltiples métodos de reproducción
Selección natural	Orientado a propósitos
Agentes cooperativos y no cooperativos	Agentes cooperativos
Benigno y maligno	Benigno
Robusto	De frágil a robusto
Envejece y muere	Normalmente no envejece
Propagación y destrucción de información	Propagación de información

Tabla 2.1: Procesos evolutivos. Información obtenida de la referencia [24]

2.3. Robótica evolutiva

El diseño, construcción y desarrollo de máquinas **autónomas** y **adaptables** es un gran problema. Existen máquinas autónomas pero no adaptables, como es el caso de la robótica industrial. En la robótica industrial se pueden encontrar robots que son autónomos, ya que repiten la misma secuencia de acciones de manera continua sin necesidad de intervención humana. La robótica autónoma está basada en el uso de sensores para detectar el espacio de trabajo y así poder controlar la tarea programada en función de las señales provenientes de los sensores. Por otra parte existen robots que se adaptan pero no son autónomos, como por ejemplo los drones. Estos drones replican las órdenes teleoperadas del operador humano que lo maneja [25].

Figura 2.4: *Izquierda: Robot industrial articulado cortesía de ABB [26]. Derecha: Drone teleoperado cortesía de Parrot [27]*

Con el objetivo de crear robots autónomos y a la vez que se sepan adaptar, la robótica gira su atención en torno a la evolución. La **robótica evolutiva** nace como una rama de la robótica y consigue que, sin detallar las actuaciónes a llevar a cabo, se varíen los parámetros adecuados para que la máquina tenga un comportamiento autónomo y adaptativo al mismo tiempo.

Debido a la naturaleza de prueba y error de los algoritmos evolutivos, se requieren una gran cantidad de evaluaciones durante la optimización. Por este motivo en la mayoría de los ejercicios de optimización robótica se hace uso en primer lugar de la simulación por ordenador. De esta forma los valores de la función de fitness pueden ser evaluados uno por uno en las simulaciones y cuando se haya alcanzado el criterio para finalizar, se puede pasar a construir y desarrollar el robot real con ese comportamiento evolutivo.

Los algoritmos evolutivos se pueden aplicar a diversas áreas de la robótica, entre otras:

- **Biorobótica evolutiva**: La biorobótica se ocupa de implementar en hardware detalles de la anatomía de un animal específico y utilizar el robot resultante como un modelo físico del animal que se quiere estudiar. En contraposición la robótica evolutiva se encarga de recrear el proceso de la evolución y como resultado se generan robots que no tienen por qué parecerse a animales existentes. La biorobótica evolutiva es una mezcla de ambos métodos: se construye un robot que se asemeja a un animal y a continuación se evoluciona un aspecto de la anatomía de dicho robot para investigar de qué manera puede evolucionar ese aspecto del animal. A modo de ejemplo, *Long* ha trabajado con la evolución de la rigidez de colas artificiales de animales nadadores y ha estudiado cómo de rápido nadan o se orientan en el agua en función de la rigidez de la cola [28].

- **Robótica cognitiva**: La robótica cognitiva es la rama de la robótica que se encarga de dotar al robot de inteligencia suficiente como para aprender y razonar cómo comportarse en situaciones complejas. Se nutre e inspira de la psicología y de la neurociencia, tomando como ejemplo la manera que tienen los niños pequeños de convertirse en adultos con complejas y crecientes capacidades.

- **Robótica de enjambres**: Este tipo de robótica trabaja con un elevado número de pequeños robots moderadamente simples con el objetivo de trabajar en equipo de cara a conseguir un objetivo común. Se estudia el diseño de dichos robots para que surjan algunos patrones de comportamiento colectivo de tal manera que se coordinen interaccionando entre ellos y con el ambiente. Este tipo de robótica toma como base los comportamientos sociales en las distintas especies animales de insectos e incluso en células. La robótica de enjambres es beneficiosa cuando designar a un robot como el líder se hace una tarea compleja debido a, por ejemplo, problemas de comunicaciones. En la Figura 2.5 puede verse un ejemplo de la cooperación entre pequeños robots de un enjambre, con un objetivo común. Incluso puede irse un paso más allá con el uso de la coevolución [29], la cual requiere competición entre grupos a la vez que cooperación entre miembros del mismo grupo. En la coevolución la capacidad de un grupo para superar a un segundo grupo incita a que el segundo grupo evolucione para defenderse del primero. Esto a su vez ejerce "presión" al primer grupo, que evolucionará hacia una nueva estrategia y así sucesivamente.

Figura 2.5: *Enjambre de robots cooperando. Proyecto "Symbrion" [30]*

- **Robótica modular**: Un robot modular está compuesto de robots indiviuales, o módulos, que pueden encadenarse o desencadenarse y de esa manera crear un robot que puede

tener una forma cambiante. En la Figura 2.6 se muestra un ejemplo de robot modular desarrollado por el doctor *Wei-Min Shen* y su equipo en el *USC Information Sciences Institute*.

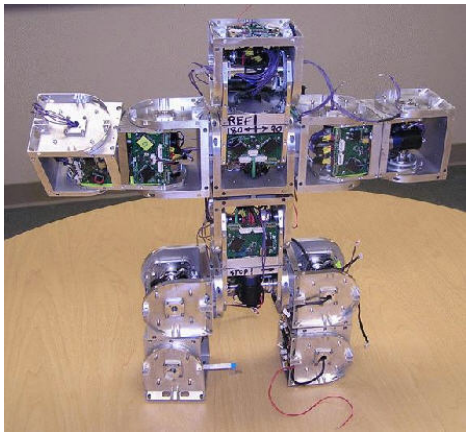


Figura 2.6: *Robot modular [31]*

Se ha demostrado que los algoritmos evolutivos pueden ser herramientas potentes para optimizar el comportamiento de un robot modular con una forma prefijada. También se ha estudiado el potencial de estos algoritmos para el montaje de estos robots modulares a partir de sus partes constituyentes [32]. Sin embargo, el desarrollo y aplicación de los algoritmos evolutivos a robots modulares continua siendo un problema abierto para la investigación.

- **Robótica blanda**: En inglés *Soft Robotics*, es una combinación de partes rígidas y partes blandas (elásticas). Normalmente los robots están compuestos de partes rígidas unidas mediante articulaciones, imitando así al esqueleto de los animales. Pero el avance en la ciencia de los materiales está haciendo posible el desarrollo de modelos que usan elementos no rígidos, saliéndose entonces de lo tradicional. De esta manera se pueden conseguir robots más flexibles y con más posibilidades de adaptación al medio. El control de esta tipología de robots no es sencillo, ya que el movimiento en una parte de robot se puede propagar a otras partes sin que se pueda prevenir. En este sentido *John Rieffel* [33] ha conseguido desarrollar un robot blando que aprovecha la propagación de movimientos en su cuerpo, en lugar de intentar evitarlos y luchar contra ellos.

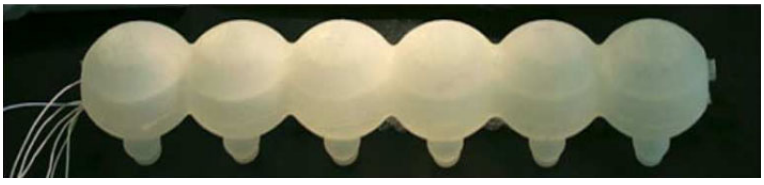


Figura 2.7: *Robot blando [33]*

2.4. Robots caminantes

El robot con el que se ha trabajado en el presente proyecto es un robot caminante cuadrúpedo [11]. Existen una gran variedad de robots móviles y entre ellos se encuentran aquellos que usan **patas** para desplazarse. Usar patas como sistema de locomoción tiene bastantes ventajas, entre las que se pueden encontrar la adaptación biológica, la adaptación al terreno en el que se desplazan y la capacidad de ser omnidireccionales (que se pueden desplazar en todas las direcciones o sentidos). Entre los robots con patas se pueden encontrar bípedos, cuadrúpedos, hexápodos, etc. Los robots hexápodos son más estables, debido a su mayor número de patas, y también más veloces. Aunque se utilizan más los cuadrúpedos por su mayor fiabilidad y menor peso [34]. El peso es tan determinante que incluso en la naturaleza no hay animal que pese más de 100 gramos y que pueda andar con 6 patas [35].

Existen en el mercado muchos robots cuadrúpedos, procediendo a continuación a nombrar algunos de ellos. El más avanzado y más famoso es el **BigDog** desarrollado por *Boston Dynamics* [36] junto con *Foster-Miller*, el *Laboratorio de Propulsión a Chorro de la Nasa* y la *Concord Field Station* de la *Universidad de Harvard* en el año 2005. Su desarrollo fue financiado por *DARPA*. Se trata de un robot con el objetivo de transportar pesadas cargas a los soldados. Además de poder llevar cargas muy pesadas puede andar, correr y escalar.

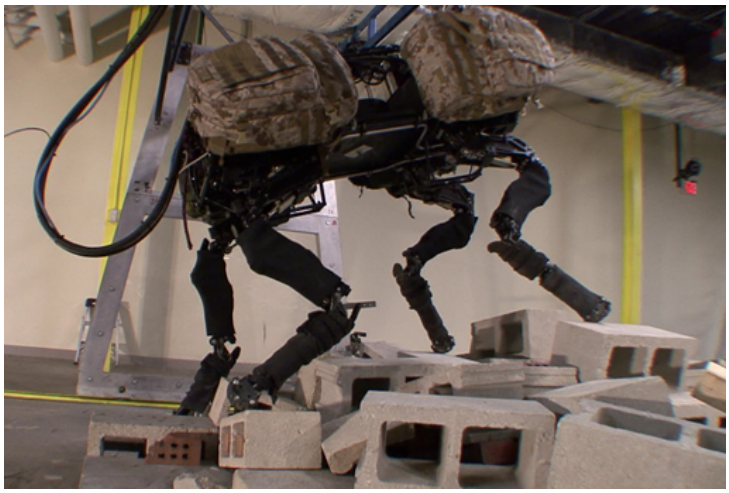


Figura 2.8: *BigDog*

Pero el **BigDog** no está solo, si no que tiene un hermano pequeño desarrollado en los mismos laboratorios y que tiene por nombre **LittleDog** [36]. Es un cuadrúpedo, al igual que su hermano mayor, aunque como su propio nombre indica tiene un tamaño menor. Está pensado para dar soporte a la investigación del aprendizaje de la locomoción y es utilizado por científicos y empresas punteras para probar sus desarrollos y avances.

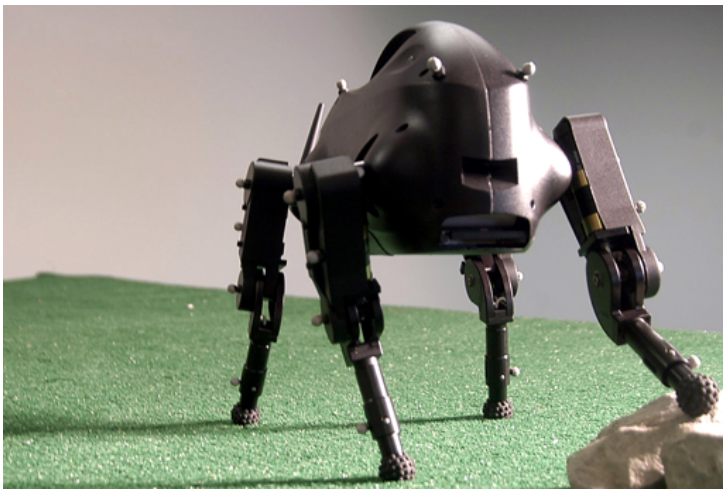


Figura 2.9: *LittleDog*

En adición a los dos robots anteriores, también desarrollado por *Boston Dynamics* y las instituciones anteriormente mencionadas, está el robot **Cheetah** y su siguiente generación llamada **WildCat** [36]. Es el robot con patas más rápido del mundo y que ha demostrado en laboratorio que es capaz de alcanzar los 40 km/h.

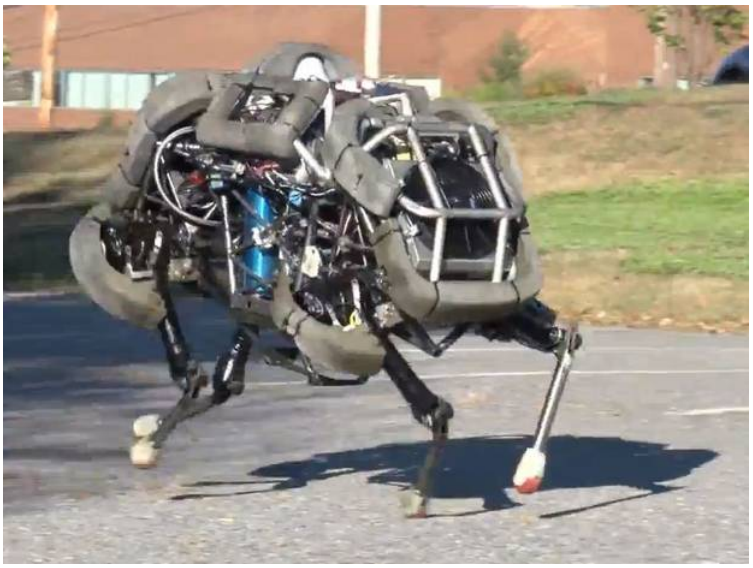


Figura 2.10: *WildCat*

Existe también cabida para robots domésticos. Los más famosos son los pertenecientes a la familia **AIBO** desarrollados por la empresa *Sony* [37], que hacen las veces de mascota. La palabra AIBO es un acrónimo que viene de las palabras "**A**rtificial **I**ntelligence Ro**bo**t" y es, además, la palabra japonesa que tiene por significado "compañero" o "amigo". En realidad

nunca fue la intención de sus creadores fabricarlos en masa, cuando se inició el proyecto de investigación en 1993. Se comenzaron a vender en 1999, conviertiéndose así en robots pioneros de su estilo. El primero de la familia fue el **ERS-111**:

Figura 2.11: *Sony AIBO® ERS-111*

El increíble éxito de la primera generación, animó a *Sony* a lanzar una segunda generación en Octubre del año 2000. Esta segunda generación se llamó **ERS-210**. Fueron mejoradas muchas funciones y además se añadieron características propuestas por los dueños del modelo anterior, como el reconocimiento de voz.

Figura 2.12: *Sony AIBO® ERS-210*

Entre los robots de su mismo estilo fue el más sofisticado en el mercado, hasta que en Noviembre de 2003 se lanzó la tercera generación. El **ERS-7** era aún más sofisticado que sus antecesores y tres versiones fueron desarrolladas durante los dos años posteriores. En Enero de 2006 *Sony* anunció el cese de la producción de la familia AIBO.

Figura 2.13: *Sony AIBO® ERS-7*

Por último, destacan también desarrollos a nivel usuario como son algunos Kits de robots cuadrúpedos. En la Figura 2.14 se muestran algunos de los Kits más famosos entre los disponibles en el mercado. El **PhantomX AX** lo desarrolla *Interbotix Labs* y puede adquirirse a través de *TrossenRobotics* [38], el **QuadRod HD** lo desarrolla *CrustCrawler Robotics* [39] y el **SQ3 Walking Robot** es un desarrollo de *LynxMotion* [40].

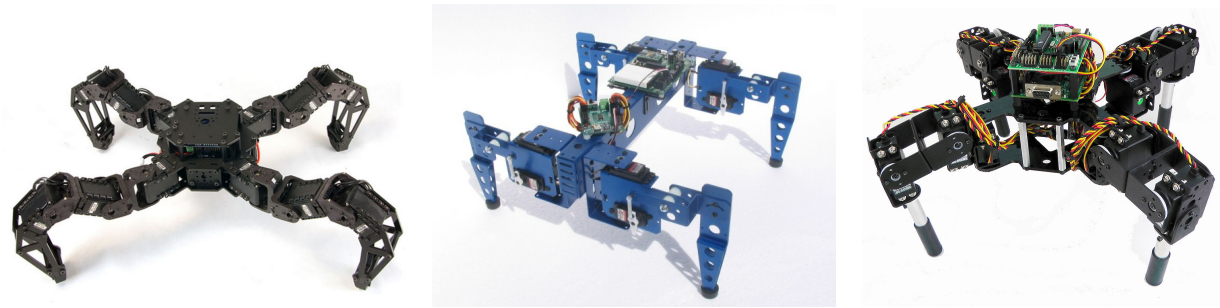


Figura 2.14: *De izquierda a derecha: PhantomX AX, QuadRod HD y SQ3U*

2.5. El Algoritmo Genético y sus componentes

Los **Algoritmos Genéticos** son una manera de optimizar problemas, como ya se ha visto con anterioridad en el Apartado 2.1. Para ello se puede caracterizar el problema y darle solución mediante la codificación en **parámetros**. Cada parámetro representará un aspecto del problema a resolver. Se pueden usar tantos parámetros como se desee, en relación a la exactitud con la que se quiera resolver el problema.

Cada individuo de la población, como se verá en el Apartado 2.5.1, representa una solución y estará codificado de manera que los parámetros van uno tras otro formando de esta manera el **cromosoma**. En el lenguaje de los Algoritmos Genéticos los parámetros se denominan **genes**. Lo normal es codificar los genes en lenguaje binario donde cada bit del gen se conoce como **alelo**. No obstante, también es posible desarrollar el Algoritmo Genético con genes no binarios, utilizando por ejemplo una codificación en genes de valores reales o enteros. En la Figura 2.15 se representa un ejemplo de codificación binaria de un individuo, en el cual se suceden los distintos parámetros o genes.

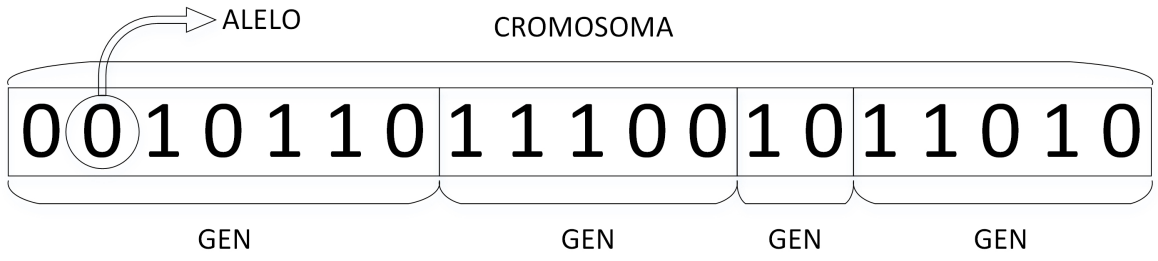


Figura 2.15: *Representación binaria de un individuo en un Algoritmo Genético*

A lo largo de este Apartado 2.5 se va a desarrollar la idea de Algoritmo Genético y se van a explicar los distintos componentes que lo constituyen. En el Apartado 2.5.1 se explica la estructura de un Algoritmo Genético, pasando después a explicar en el Apartado 2.5.2 los distintos operadores genéticos que se utilizan. Finalmente, en el Apartado 2.5.3, se trata la función de *fitness* utilizada para saber la bondad de los individuos.

2.5.1. Algoritmo principal

Podría decirse que el **algoritmo principal** es la base sobre la que trabaja el genético. En el algoritmo principal queda reflejado el funcionamiento del Algoritmo Genético, usándose para ello los operadores genéticos y la función de bondad. Para entender el funcionamiento del algoritmo principal se hace necesario primero repasar los conceptos de población y de individuo:

- **Individuo**: Representación de una posible solución al problema. A cada individuo se le asocia una medida de bondad de la solución, que se conoce como *fitness*.
- **Población**: Conjunto de individuos con los que trabaja el Algoritmo Genético para solucionar el problema que se quiere optimizar.

Utilizando los conceptos explicados hasta el momento, podría desarrollarse un primer diagrama de flujo sencillo del funcionamiento del algoritmo; como puede verse a continuación en la Figura 2.16:

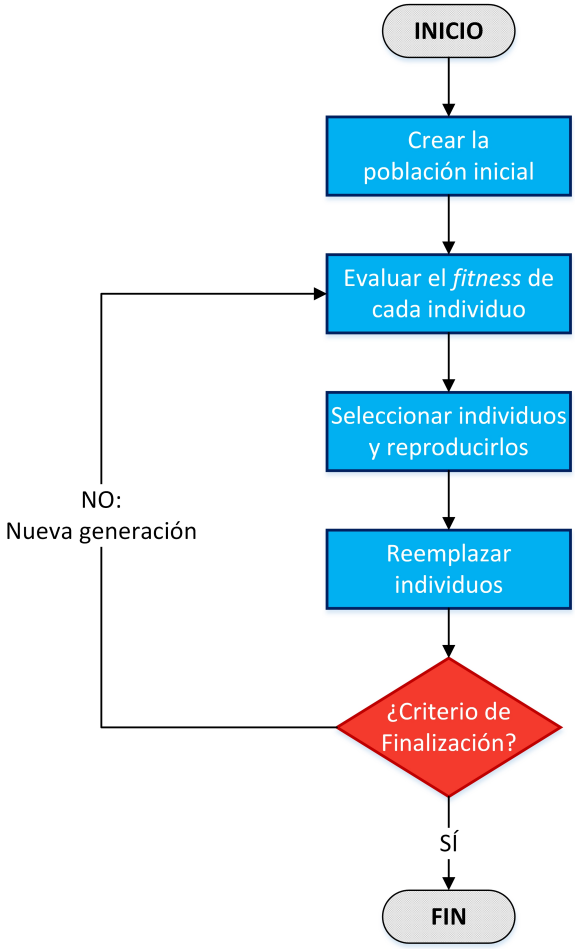


Figura 2.16: *Diagrama de flujo sencillo de un Algoritmo Genético*

El inicio de los Algoritmos Genéticos, como ya se comentó en el Apartado 2.1, fue en parte gracias a *Holland* [16]. Del funcionamiento genérico propuesto por *Holland*, han surgido numerosas variaciones a lo largo del tiempo. Más adelante se comentarán algunas de ellas. En la Figura 2.17 puede verse un diagrama de flujo del funcionamiento genérico de un Algoritmo Genético, más desarrollado que la primera aproximación de la Figura 2.16.

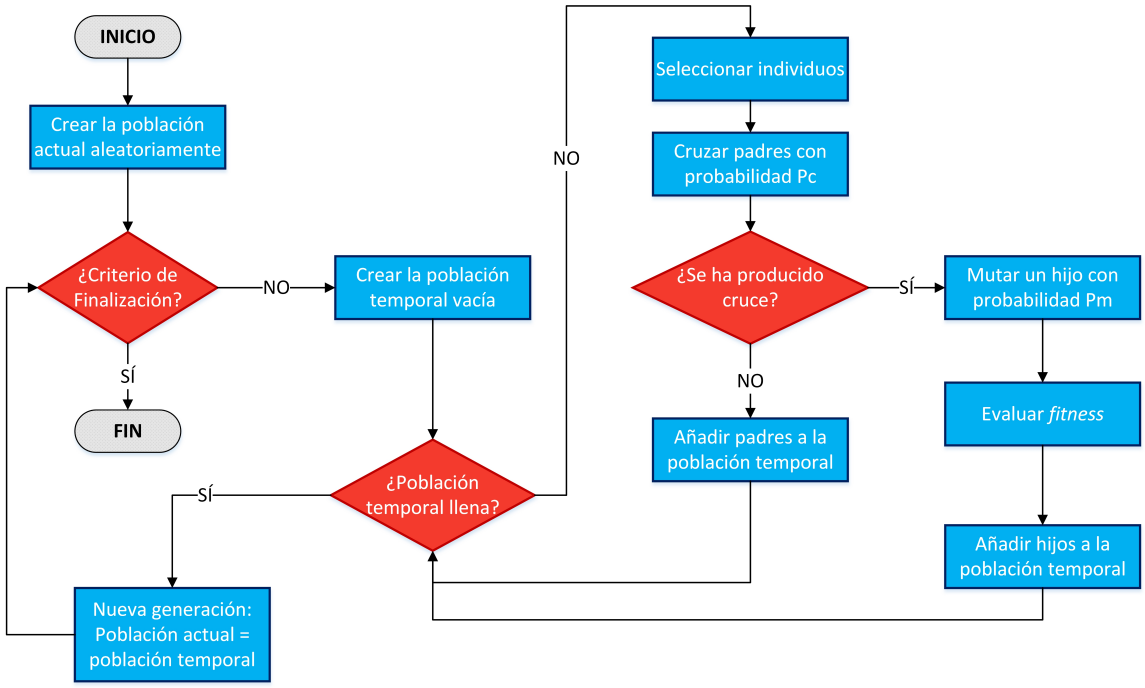


Figura 2.17: *Diagrama de flujo genérico de un Algoritmo Genético*

La **población inicial** está compuesta por individuos generados de manera aleatoria. De esta forma, estos individuos irán evolucionando con el paso de las generaciones. El uso de una **población actual** y una **población temporal** está justificado, ya que de esta manera se puede ir generando descendientes sin "pisar" la población actual. Posteriormente, una vez la población temporal esté llena, se puede proceder a igualar la población actual con la temporal. La población temporal será vaciada y vuelta a llenar con individuos nuevos en la siguiente generación.

Para el paso de una generación a otra, se utilizan los operadores genéticos que se verán con más detalle en el Apartado 2.5.2. A grandes rasgos, cuando se escogen dos individuos se estudia la propabilidad de cruce **Pc**, que normalmente está en torno al 90 %, y pueden pasar dos cosas:

1. **Cruce sí**: Se produce el cruce (ver Apartado 2.5.2.2), siguiendo una reproducción de tipo sexual en la que los genes de los dos individuos seleccionados se combinan para generar un descendiente. Tras el cruce, se evalúa la probabilidad de mutación del descendiente **Pm** (que normalmente no es mayor del 2 %, ver Apartado 2.5.2.4). Este modo de proceder imita a lo que ocurre en realidad en la naturaleza, cuando se produce algún error en la reproducción de los genes. Por último, se evalúa el *fitness* (ver Apartado 2.5.3) del individuo generado antes de introducirlo en la población.
2. **Cruce no**: Hay una baja probabilidad de que el cruce no llegue a producirse. Lo que ocurre entonces, es que se copian los dos individuos seleccionados sin variaciones directamente a la población (ver Apartado 2.5.2.5), siguiendo una reproducción de tipo asexual.

Existen varios motivos para alcanzar el **fin del algoritmo**. Entre otros, puede ser porque se haya alcanzado el número de generaciones máximo prefijado, porque la población converja hacia una solución o porque los mejores individuos de la población tengan un *fitness* suficientemente elevado como para deducir que son buenas soluciones del problema y no es necesario entonces avanzar más generaciones. Parece evidente que el mayor deseo es que se alcance el final del algoritmo debido a la última razón expuesta, puesto que eso significaría que el proceso ha sido exitoso.

Sin embargo, existen varias alternativas a este tipo de Algoritmo Genético genérico. Es necesario valorar detenidamente el problema y decidir todos los detalles con los que se construirá el algoritmo. Algunas de las variantes que podrían plantearse son las siguientes [41]:

- **Elitismo**: Con el objetivo de evitar la pérdida de los mejores individuos de cada generación, se copian estos mejores individuos siempre en la población temporal.
- **Mutación aleatoria**: La variación consiste en que en lugar de mutar al descendiente procedente de un cruce, como se hace en el modelo genérico, se elige de manera aleatoria un individuo de entre todos los que componen la población.
- **Reemplazo**: Es quizás la variante más utilizada. Consiste en no hacer uso de una población temporal. En este caso se aplicarán los operadores genéticos directamente sobre la población actual y cuando se genere un descendiente no será incluido en la población directamente. Puesto que el tamaño de la población no puede aumentar, lo que se hace es suprimir algún individuo de la población actual para dejar un hueco para insertar el nuevo descendiente generado. Para realizar esta operación se hace uso de los algoritmos de reemplazo, que se verán en el Apartado 2.5.2.6. En esta variación el paso de una generación a otra no se realiza cuando se llene la población temporal (puesto que no existe tal población), si no que se aumenta una generación cuando hayan sido realizados un cierto número de cruces y mutaciones. El número límite de cruces y mutaciones queda definido en función de Pc, Pm y el tamaño de la población.

2.5.2. Operadores genéticos

2.5.2.1. Selección

Mediante los métodos de selección se eligen los individuos que van a reproducirse y generar descendencia a la próxima generación. La selección intentará imitar a la selección natural, por lo que los individuos más aptos tendrán más probabilidades de ser seleccionados para reproducirse. Un uso extendido de estos métodos es elegir un individuo por selección y el otro de manera aleatoria. Por otra parte, no es aconsejable no proporcionar ninguna opción de reproducción a los individuos menos preparados de la población ya que podría correrse el riesgo de que la población convergiese al cabo de pocas generaciones.

Los métodos de selección por ruleta y por torneo aquí expuestos, son de tipo probabilístico. Existen otros métodos de tipo determinístico que pueden evitar problemas de predominancia de algunos individuos [42] [43] [44]. En ellos se otorga a cada individuo un número de veces en las que se puede reproducir, en relación a su aptitud reflejada en la función de *fitness*.

- **Selección por ruleta**:

  También llamada en muchas publicaciones como "*Selección de Montecarlo*", es un método de selección muy utilizado cuando la población no tiene un tamaño demasiado grande. La idea consiste en asignar a cada individuo de la población una porción de ruleta. El tamaño de la porción asignada a cada individuo será una consecuencia directa de su función de aptitud o *fitness*. De esta manera, la asignación de una porción más grande será para los individuos más aptos y los menos aptos tendrán, por tanto, una porción de ruleta más pequeña.

  Debido a que normalmente los individuos están ordenados según su *fitness*, los individuos más aptos estarán al principio de la ruleta con sus porciones grandes y los individuos menos aptos estarán al final de la ruleta con sus porciones más pequeñas. El tamaño de la porción de ruleta de cada individuo puede ponerse como un número proporcional entre 0 y 1. El procedimiento para seleccionar un individuo puede ser en primer lugar generar un número aleatorio en el intervalo [0,1] y en segundo lugar ir sumando las proporciones de cada individuo hasta que se sobrepase el número aleatorio generado. De esta manera se puede detectar el individuo escogido por la ruleta.

- **Selección por torneo**:

  Como su propio nombre indica, lo que busca este método es realizar torneos entre individuos. Esto es, comparar de manera directa un individuo con otro. La idea es enfrentar "n" individuos y escoger al mejor de ellos en función del *fitness* de cada uno. Un valor usual suele ser "n=2".

  No obstante, "n" puede tomar valores más grandes. Si el valor de "n" es muy grande, lo que ocurre es que se enfrentan muchos individuos a la vez y por tanto la presión de selección será más elevada que en los casos en que "n" sea pequeño. Cuando se tiene una presión de selección elevada ocurre que los individuos con un *fitness* bajo, es decir aquellos individuos menos preparados, no dispondrán apenas de oportunidades de reproducción.

2.5.2.2. Cruce

El mecanismo de cruce se realiza con una probabilidad de cruce *Pc*, que en la mayoría de los casos se sitúa en torno al 90 %. En el cruce se mezclan los genes de los progenitores y dependiendo del método de cruce esta mezcla se hará de una u otra manera. El concepto de cruce hace suponer que si se reproducen dos buenos individuos, sus descendientes tendrán una buena mezcla de sus genes y por tanto tendrán un *fitness* mejor. Sin embargo, puede ocurrir que

de dos individuos buenos surja algún descendiente cuyos genes sean una mezcla de los peores genes de sus padres. De manera que, en tal caso, el descendiente no tendría por qué ser mejor que sus progenitores.

Existen multitud de tipos de cruces, no siendo de excesiva complicación incluso pensar en métodos nuevos. Sin embargo, solo se expondrán a continuación los más utilizados.

- **Cruce en 1 punto (SPX)**:

  En inglés *Single Point Crossover*, es el primer método de cruce en el que se puede pensar y es el más simple. Consiste en cortar los cromosomas de los individuos por un único punto, seleccionado previamente de manera aleatoria. Se debe tener cuidado de que el punto seleccionado no sea un borde de cromosoma, porque de lo contrario no se produciría ninguna división del mismo. El resultado es que cada cromosoma progenitor tiene ahora dos partes que se podrán mezclar entre ellas para formar la descendencia. De este proceso pueden ser obtenidos dos descendientes, según la combinanción (ver Figura 2.18), y puede que haya que decidir con cual de ellos quedarse.

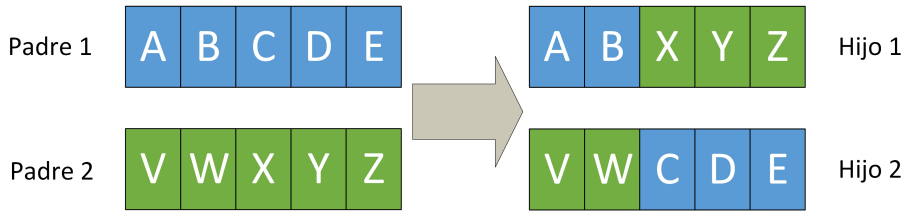


Figura 2.18: *Cruce en 1 punto (SPX)*

- **Cruce en 2 puntos (DPX)**:

  En inglés *Double Point Crossover*, es la ampliación del caso de corte por un punto. En este método el corte se realiza por dos puntos seleccionados previamente de manera aleatoria, de tal manera que resultan tres segmentos del cromosoma. Igualmente se debe tener cuidado de que los puntos seleccionados no sean un borde de cromosoma, porque de lo contrario no se producirían tres segmentos, sino dos. De este proceso pueden ser obtenidos dos descendientes, según la combinanción (ver Figura 2.19), y puede que haya que decidir con cual de ellos quedarse.

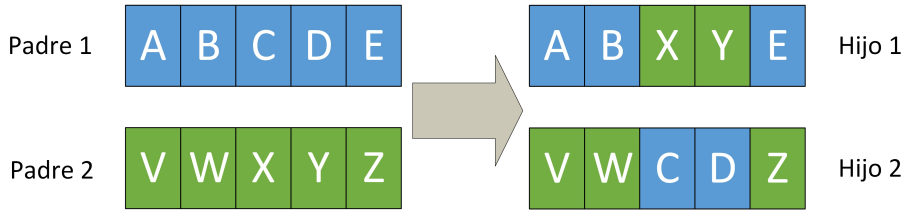


Figura 2.19: *Cruce en 2 puntos (DPX)*

- **Cruce en "p" puntos**:

  Se pueden generalizar los dos métodos anteriores y realizar un corte en "p" puntos. Sin embargo, hay estudios como el de *Kenneth Alan De Jong* [45] que demuestran que generar más puntos de corte no ofrece beneficio con respecto al SPX o al DPX. Esto se debe a que es más sencillo perder la esencia del cromosoma de un individuo puesto que se está diviendo en demasiadas partes y posiblemente se esté perdiendo la parte del cromosoma que genera la bondad del progenitor. Por otro lado, se admite que el DPX ofrece una mejor solución que el SPX.

- **Cruce uniforme (UPX)**:

  En inglés *Uniform Point Crossover*, es un método en el que no se generan puntos de corte del cromosoma, si no que se genera una máscara de cruce uniforme. La máscara de cruce será generada de manera aleatoria y será binaria. Si en una posición de la máscara hay un 1, se copiará el gen correspondiente de uno de los padres y si, por el contrario, hay un 0 se copiará el gen correspondiente del otro padre. De este proceso pueden ser obtenidos dos descendientes, según la interpretación de los 0 y 1 de la máscara, y puede que haya que decidir con cual de ellos quedarse. En la Figura 2.20 se muestra el esquema de generación de un hijo por cruce con máscara, habiendo un segundo hijo que será inverso al primero.

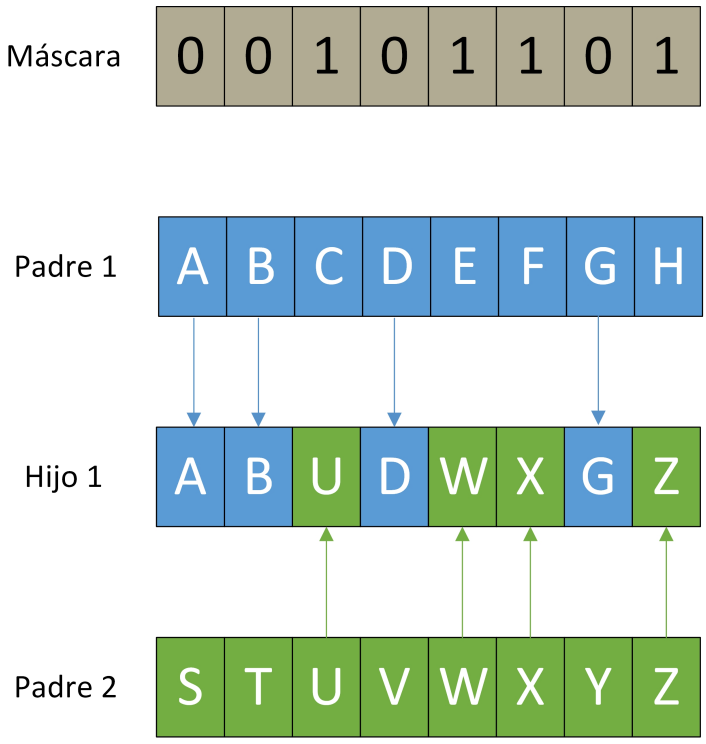


Figura 2.20: *Primer hijo producido por un cruce uniforme (UPX)*

2.5.2.3. Cruces de codificaciones no binarias

Los métodos de cruce vistos en el Apartado 2.5.2.2 pueden servir para problemas que tengan codificaciones binarias y también problemas que tengan codificaciones no binarias (reales o enteros). De tal manera que se puede utilizar, por ejemplo, el cruce en 1 punto tanto en una codificación binaria como en una codificación real.

No obstante, existen algunos métodos de cruce especialmente útiles para codificaciones no binarias [46]. Un ejemplo de ellos que se nos podría ocurrir rápidamente es la media, en la que cada gen del individuo hijo sería el valor medio de los genes correspondientes de los progenitores.

Si se parte de los cromosomas $C_1 = (c_1^1 ... c_n^1)$ y $C_2 = (c_1^2 ... c_n^2)$ de dos individuos progenitores que se quieren reproducir, dependiendo del método de cruce para codificaciones no binarias utilizado, se obtendrán diferentes efectos en el hijo resultante. A continuación se describen algunos métodos, no siendo los únicos existentes.

- **Cruce plano** [47]

  En inglés *Flat Crossover*, el hijo $H = (h_1, ..., h_i, ..., h_n)$ resulta de elegir cada gen h_i de manera aleatoria en el intervalo definido por los genes correspondientes de los progenitores $[c_i^1, c_i^2]$.

- **Cruce BLX-α** [48]

  En inglés *BLX-α Crossover*, es un caso genérico del anterior ya que el BLX-0.0 es equivalente al cruce plano. El hijo $H = (h_1, ..., h_i, ..., h_n)$ resulta de elegir cada gen h_i de manera aleatoria en el intervalo $[c_{min} - I\alpha, c_{max} + I\alpha]$, siendo $I = c_{max} - c_{min}$, $c_{max} = max(c_i^1, c_i^2)$ y $c_{min} = min(c_i^1, c_i^2)$.

- **Cruce aritmético** [49]

  En inglés *Arithmetical Crossover*, se pueden generar dos hijos $H_1 = (h_1^1, ..., h_i^1, ..., h_n^1)$ y $H_2 = (h_1^2, ..., h_i^2, ..., h_n^2)$. Debido a que se puede crear más de un hijo, es posible que haya que elegir entre ellos. En dichos hijos cada gen tiene un valor de $h_i^1 = \lambda c_i^1 + (1 - \lambda) c_i^2$ y $h_i^2 = \lambda c_i^2 + (1 - \lambda) c_i^1$, siendo λ una constante con respecto al número de generaciones que han pasado.

- **Cruce lineal** [50]

  En inglés *Linear Crossover*, se pueden generar tres hijos $H_1 = (h_1^1, ..., h_i^1, ..., h_n^1)$, $H_2 = (h_1^2, ..., h_i^2, ..., h_n^2)$ y $H_3 = (h_1^3, ..., h_i^3, ..., h_n^3)$. Debido a que se puede crear más de un hijo, es posible que haya que elegir entre ellos. En dichos hijos cada gen tiene un valor de $h_i^1 = \frac{1}{2} c_i^1 + \frac{1}{2} c_i^2$, $h_i^2 = \frac{3}{2} c_i^1 - \frac{1}{2} c_i^2$ y $h_i^3 = -\frac{1}{2} c_i^1 + \frac{3}{2} c_i^2$.

- **Cruce discreto** [51]

  En inglés *Discrete Crossover*, el hijo $H = (h_1, ..., h_i, ..., h_n)$ resulta de elegir cada gen h_i de manera aleatoria entre los valores de los genes correspondientes de los progenitores c_i^1 y c_i^2.

- **Cruce extendido** [51]

  En inglés *Extended Crossover*, el hijo $H = (h_1, ..., h_i, ..., h_n)$ resulta de elegir cada gen $h_i = c_i^1 + \alpha(c_i^2 - c_i^1)$, siendo α una constante dentro del intervalo $[-\frac{1}{4}, \frac{5}{4}]$.

- **Cruce heurístico de *Wright*** [52]

  En inglés *Wright's Heuristic Crossover*. Suponiendo que C_1 es el progenitor con el mejor *fitness* de los dos, entonces cada gen h_i del hijo resultante $H = (h_1, ..., h_i, ..., h_n)$ tiene un valor de $h_i = r(c_i^1 - c_i^2) + c_i^1$, siendo r una constante dentro del intervalo $[0, 1]$.

En la Tabla 2.2 se recogen, de manera ilustrativa, las representaciones de algunos de los cruces anteriormente citados:

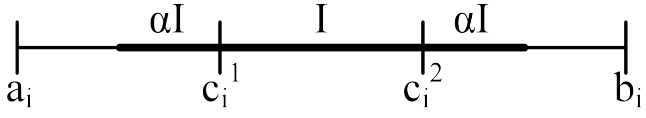
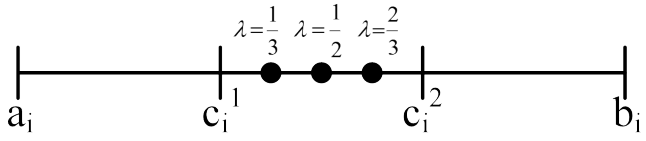
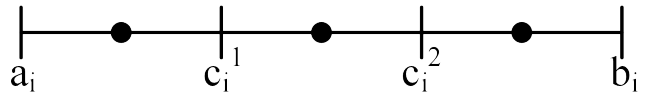
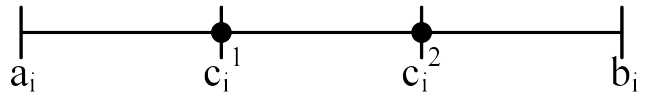
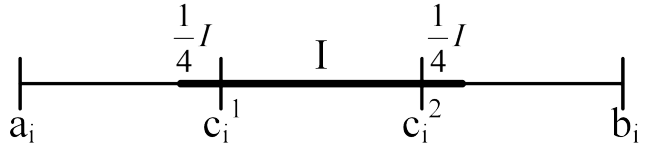
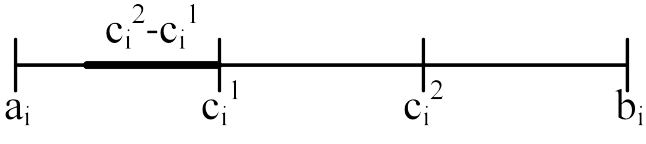
BLX-α	
Aritmético	
Lineal	
Discreto	
Extendido	
Heurístico de *Wright*	

Tabla 2.2: Operadores de cruce de codificaciones no binarias. Información obtenida de la referencia [46]

2.5.2.4. Mutación

La probabilidad de mutación *Pm* suele ser baja, generalmente menor del 1 %. Aunque, en algunos casos puede llegar hasta el 2 %, no siendo normal sobrepasar este valor. Esto se debe a que el valor de bondad de un individuo se reduce al producirse una mutación. La mutación consiste en la modificación aleatoria de uno o más genes del cromosoma del individuo (normalmente uno). En la Figura 2.17 se muestra que, de manera general, se suele producir la mutación tras el cruce. Este modo de proceder imita a lo que ocurre en realidad en la naturaleza. En la naturaleza la mutación se produce tras la reproducción de dos individuos: a la hora de cruzar sus cromosomas se produce algún tipo de error que da lugar a la mutación de algún gen del cromosoma del individuo descendiente del cruce.

No obstante en un Algoritmo Genético, se puede realizar también la mutación seleccionando un individuo aleatorio de la población actual y mutándolo previa inserción a la nueva población. Aunque este modo de mutar individuos tiene el inconveniente de que no se asemeja al comportamiento de la selección natural, que es lo que persigue al fin y al cabo el Algoritmo Genético.

En el caso de tener una codificación binaria, lo más frecuente suele ser cambiar un gen del cromosoma (elegido de manera aleatoria). Esto es, si el gen vale 1 pasará a valer 0. Si por el contrario el gen vale 0, pasará a valer 1. En la Figura 2.21 se ilustra un ejemplo de este tipo de mutación. Otros modos de mutar en los que se puede pensar es intercambiar los valores del gen escogido aleatoriamente y el siguiente, o intercambiar los valores de dos genes aleatorios del cromosoma.

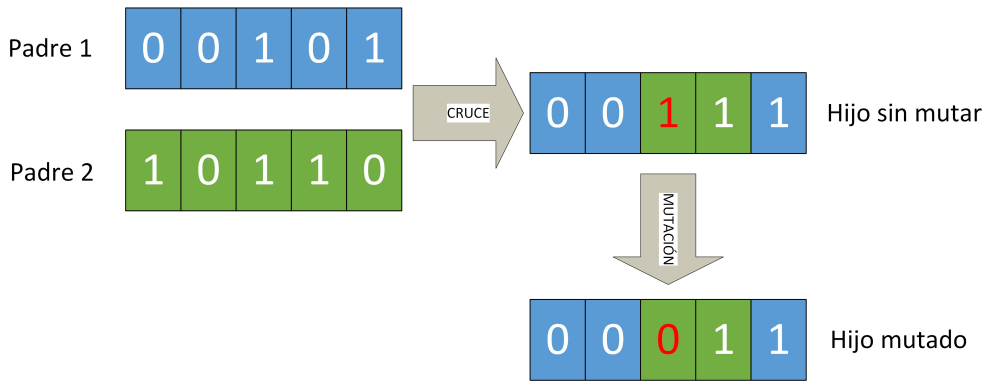


Figura 2.21: *Ejemplo de mutación*

En los casos en los que se tengan codificaciones que no sean binarias, pueden utilizarse incluso más tipos de mutación. Algunos de estos tipos pueden ser incrementar o decrementar el valor de un gen una cierta cantidad aleatoria, sumar o restar el valor de un gen a otro, multiplicar un gen por un número aleatorio dentro del rango [0,1], etc. Por tanto, pueden pensarse muchas maneras de mutar un individuo y queda abierta la elección sobre qué tipo de mutación utilizar.

2.5.2.5. Copia

La copia es el otro tipo de reproducción además del cruce. En este caso la reproducción es de tipo asexual. La copia se produce cuando, como consecuencia de la probabilidad *Pc*, no se ha producido cruce. Dado que la probabilidad de cruce es bastante elevada en la mayoría de los casos, el método de reproducción por copia no es muy habitual que llegue a darse. Si la copia fuera muy frecuente se correría el riesgo de que la población convergiera de manera rápida hacia un individuo en concreto.

La copia es un método de reproducción mucho más simple que el cruce. Consiste en copiar directamente los progenitores en la nueva población. Por tanto, los individuos progenitores pasarán sin modificación alguna a la siguiente generación.

2.5.2.6. Reemplazo

En la Figura 2.17 se ha mostrado el esquema del funcionamiento de un Algoritmo Genético genérico. No obstante, como ya se ha explicado, pueden existir variaciones de dicha forma de proceder genérica. Una de las variaciones de las que se ha hablado es la de no usar población temporal, realizando la selección y los cruces directamente en la población actual. Para ello es necesario hacer uso de los mecanismos de reemplazo, ya que los individuos se seleccionan directamente de la población actual y, tras el cruce, es necesario suprimir un individuo de la población para introducir el nuevo descendiente creado. Para llevar a cabo el reemplazo de un individuo de la población por el nuevo descendiente, hay varias maneras de elegir el individuo a eliminar.

- **Reemplazo de los padres**: Se selecciona uno de los padres para hacer hueco a su hijo. La selección del padre a eliminar, entre los dos existentes, puede ser aleatoria o en función de la bondad de éstos. Si se hace uso de la bondad de los padres, lo normal es eliminar al menos preparado de los dos.
- **Reemplazo de un individuo aleatorio**: El individuo a suprimir se elige de manera aleatoria entre todos los que componen la población.
- **Reemplazo de un individuo similar**: Se puede optar por reemplazar a un individuo que tenga unas características similares al nuevo descendiente que se va a insertar. Para ello se hace uso de la bondad de los individuos para encontrar alguno que tenga un *fitness* parecido al que tiene el nuevo individuo.
- **Reemplazo de los peores individuos**: En este caso el hueco se realiza de entre los peores individuos de la población, en relación a su *fitness*. De esta forma, a no ser que el nuevo individuo sea muy "malo", el Algoritmo Genético siempre va a evolucionar hacia mejor ya que los individuos menos preparados son sustituidos.

2.5.3. Función de *fitness*

Al igual que ocurre con la codificación de los individuos, en cada problema habrá que utilizar una función de bondad que se adapte a dicho problema. Es necesario, para el correcto funcionamiento del Algortimo Genético, disponer de un mecanismo que refleje cómo de buena es la solución que representa cada individuo.

La función de bondad (también llamada función de *fitness*) refleja, por tanto, la adecuación de un individuo al problema. Se asigna un valor numérico a dicha bondad, que recibe el nombre de **ajuste**. Si se piensa en lo que ocurre en la naturaleza, podría decirse que el ajuste representa las posibilidades de un individuo de llegar a la edad adulta y reproducirse. Evidentemente esas posibilidades serán función de la adecuación de dicho individuo al medio en cuanto a lucha por los recursos se refiere.

En el Algoritmo Genético habrá que evaluar la bondad del individuo en la consecución de lo que requiera el problema y, en función de esa evaluación de la adecuación, asignar un valor de ajuste. La evaluación de la aptitud de un individuo será, en general, distinta para cada problema y deberá pensarse la manera de evaluar dicha aptitud. Los individuos son ordenados por el valor de ajuste de cada uno, de tal forma que los mejores individuos tendrán un valor de ajuste alto si el problema es de maximización. Si por el contrario el problema es de minimización, los individuos más aptos tendrán un valor de ajuste más bajo que los menos aptos.

Hay muchos tipos de fitness, pero solo se van a mencionar aquí algunos de ellos. Según *John R. Koza* los cuatro tipos de *fitness* más utilizados son los siguientes [53]:

1. **Fitness puro** $r(i,t)$: Es la medida de ajuste establecida en la propia terminología natural del problema. En la Ecuación (2.1) se muestra el cálculo del *fitness puro* de un individuo *i* en una generación *t*. En dicha ecuación *s(i,j)* es el valor deseado, *c(i,j)* es el valor obtenido y *N* es el número de casos.

$$r(i,t) = \sum_{j=1}^{N} |s(i,j) - c(i,j)| \tag{2.1}$$

   Un ejemplo podría ser el número de piezas de comida que lleva una hormiga al hormiguero, siendo mejor cuantas más piezas consiga la hormiga en cuestión. En los casos de maximización, como el de las hormigas, serán mejores los individuos con un *fitness puro* elevado. Mientras que los individuos con un *fitness puro* más bajo son menos interesantes.

2. **Fitness estandarizado** $s(i,t)$: Si se desea diferenciar entre los casos de maximización y de minimización se puede partir de la Ecuación (2.1) para llegar a la Ecuación (2.2), que es la ecuación del *fitness estandarizado*. En dicha ecuación r_{max} es una cota superior.

$$s(i,t) = \begin{cases} r(i,t) & \text{minimización,} \\ r_{max} - r(i,t) & \text{maximización} \end{cases} \tag{2.2}$$

3. **Fitness ajustado** $a(i,t)$: Si se utiliza la Ecuación (2.3) se obtiene el *fitness ajustado*, que se encuentra en el intervalo [0,1].

$$a(i,t) = \frac{1}{1+s(i,t)} \tag{2.3}$$

4. **Fitness normalizado** $n(i,t)$: Introduce un concepto que no aparece en los tres casos anteriores. La idea es referir el *fitness* de un individuo con respecto al resto de soluciones de la población. El *fitness normalizado* se encuentra en el intervalo [0,1] y su Ecuación es la siguiente (2.4).

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{N} a(k,t)} \tag{2.4}$$

   De esta forma, si un individuo tiene un *fitness normalizado* cercano a 1, además de indicar que es un individuo muy bueno, indica que el individuo representa una solución notablemente mejor que las del resto de la población.

3

Entorno de trabajo

3.1. Entorno de simulación. V-REP

Existen numerosos programas de ordenador que permiten simular robots. De entre todos se ha escogido **V-REP** [54], desarrollado por **Coppelia Robotics**, por varios motivos:

- El autor realizó una práctica sobre dicho programa en la asignatura de *Robótica*, en la que pudo aprender los controles y funcionalidades básicas del programa.
- El entorno está bastante elaborado de manera que, tal y como el propio eslogan del programa reza, se puede crear, componer y simular cualquier robot. Dispone de 3 motores de físicas: Bullet Physics, ODE y Vortex Dynamics; por lo que es posible customizar la física del modelo de una manera cómoda.
- Como se verá en el Apartado 3.3, el entorno de programación utilizado es **Matlab**. Teniendo en cuenta que V-REP cuenta con una API (del inglés *Application Programming Interface*) desde Matlab bastante extensa, toma importancia el uso de V-REP para simular el robot.

El nombre del programa, **V-REP**, es un acrónimo de *"**V**irtual **R**obot **E**xperimentation **P**latform"*, que en español podría traducirse como "Plataforma de Experimentación Virtual de Robots". En V-REP es posible controlar cada objeto y cada modelo de la escena de manera individual desde Matlab mediante el uso de la API remota. Dicha API pone a disposición del usuario el uso de cerca de 100 funciones con las que controlar la simulación.

A su vez es posible importar y exportar de manera sencilla objetos, además de poder especificar colores y texturas para disfrutar de una mejor visualización. Igualmente dispone de elaborados menús y de una interacción total incluso durante las simulaciones, permitiendo de esta forma incluso cambiar el ángulo y posición de visión mientras la simulación se encuentra en proceso.

En adición a todo lo anterior, V-REP dispone de un "Manual de usuario" [55] en el cual se recoge todo lo que se necesita saber sobre el programa. Se incluyen también numerosos tutoriales para aprender a utilizar distintas funcionalidades como, por ejemplo, importar y preparar cuerpos rígidos, utilizar controladores externos, desarrollar un seguidor de línea o una cinta transportadora, etc.

Finalmente cabe destacar la existencia de un foro en inglés [56] con una extensa comunidad. En el foro es posible plantear dudas o problemas, obteniendo normalmente respuesta o ayuda

con rapidez. El foro es mantenido por el propio equipo de *Coppelia Robotics* y es habitual que ellos mismos respondan en los hilos que crean los usuarios, brindando así ayuda de primera mano. Para el desarrollo del presente proyecto, el descubrimiento y uso de dicho foro ha sido crucial para la superación de los problemas que se presentaban.

En la Figura 3.1 se muestra el entorno de simulación de V-REP, en la que también puede observarse el modelo del robot que se ha desarrollado.

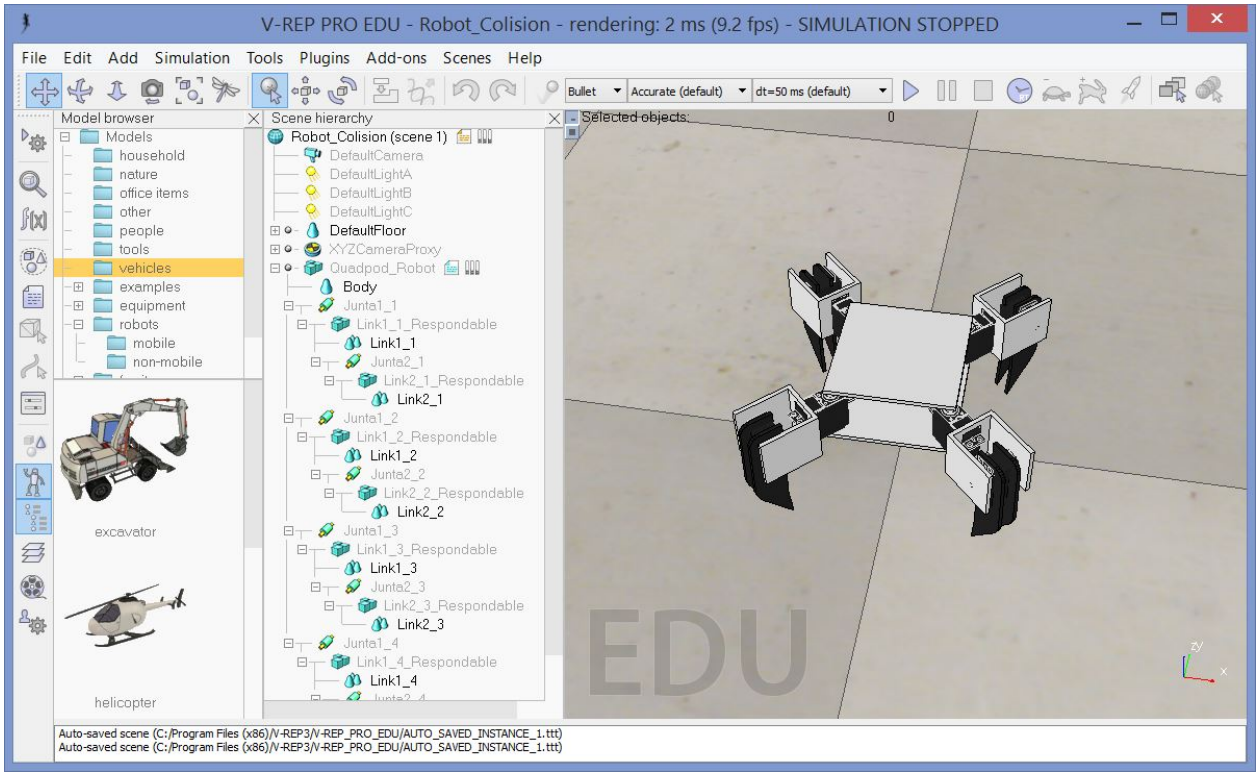


Figura 3.1: *Ventana de V-REP con el modelo desarrollado del robot*

3.2. Modelo de simulación

Para la creación de un modelo del robot real con el que poder realizar las simulaciones, se han tomado medidas del robot real y además se han consultado los planos proporcionados por la referencia [11]. Dichos planos se han incluido a su vez en el Anexo I. En el Apartado 3.2.1 se describe el diseño de cada una de las piezas del modelo y en el Apartado 3.2.2 se describe el proceso seguido para ensamblar todas las piezas y así montar el modelo de simulación en V-REP.

3.2.1. Diseño de piezas

Cada pieza del robot, exceptuando los motores, se han diseñado en **Autocad**. El robot es un cuadrúpedo donde cada pata tiene dos grados de libertad y, por tanto, dos motores. Además de los dos motores, cada pata está compuesta de dos partes que se han denominado codo y pie. En la Figura 3.2 se muestra la estructura de las patas del robot.

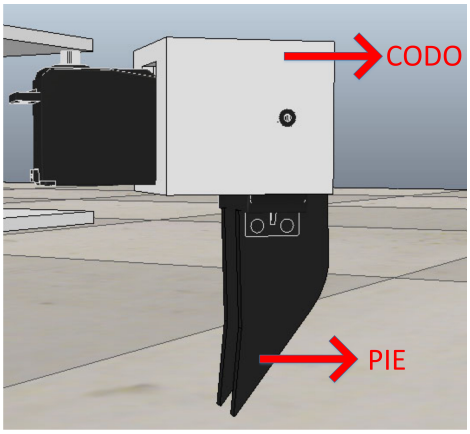


Figura 3.2: *Estructura de una pata del robot*

El diseño en *Autocad* de ambas partes de la pata, se muestra en la figura 3.3. A la izquierda se encuentra el codo, con el orificio pertinente para poder ensamblarlo con el motor, y a la derecha se encuentra el pie. Nótese que el pie está compuesto de dos piezas iguales a la mostrada con una separación entre ellas, pero en la figura solo se muestra una pieza de las dos que componen el pie con el objetivo de poder apreciar mejor su forma.

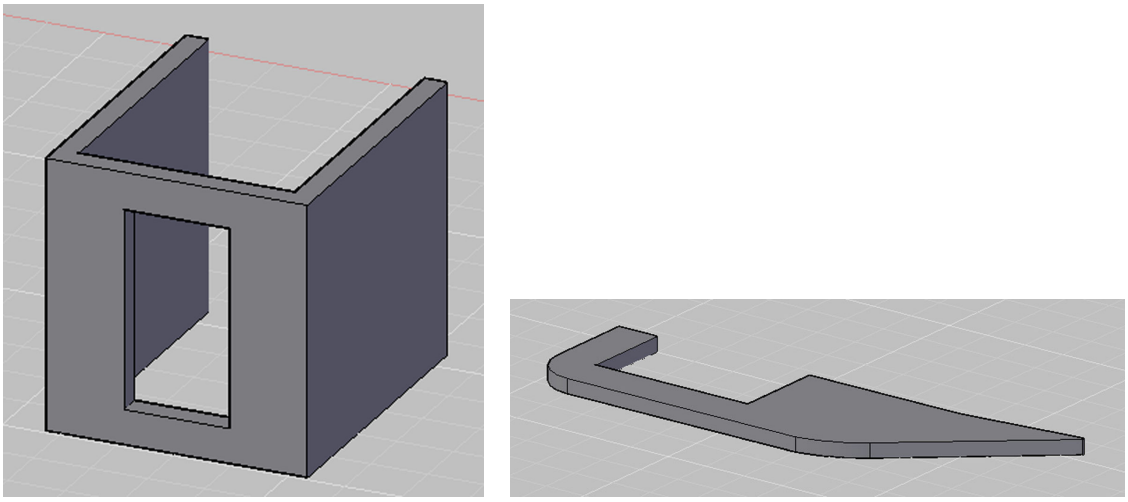


Figura 3.3: *Diseño en Autocad del codo y el pie del robot*

El cuerpo del robot son dos placas en las que se encuentran las cuatro patas en cada una de las esquinas y entre las que, en el robot real, se encuentra la circuitería de control. Para el modelo de simulación, por cuestiones de simplicidad, ambas placas se han diseñado perfectamente cuadradas. El error con respecto al diseño original, que se puede consultar en el Anexo I, es despreciable debido al poco material de diferencia. A su vez la simetría del modelo no se ve afectada, por lo que es el modelo cuadrado el que se ha diseñado en *Autocad* como puede verse en la Figura 3.4.

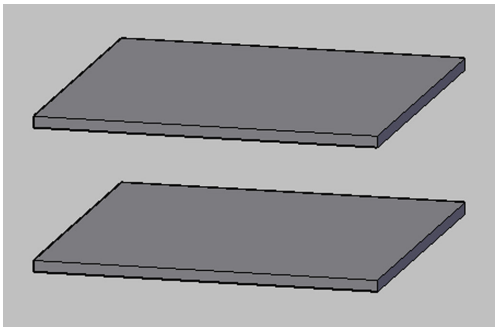


Figura 3.4: *Diseño en Autocad del cuerpo del robot*

En cuanto a los motores, el robot real lleva incorporados servos de la marca *Futaba* y modelo *3003*. Debido a la complejidad del diseño CAD de dichos servos, se ha recurrido a un diseño CAD de uso libre desarrollado por *Juan González* [57]. En la Figura 3.5 se muestra el diseño de los servos.

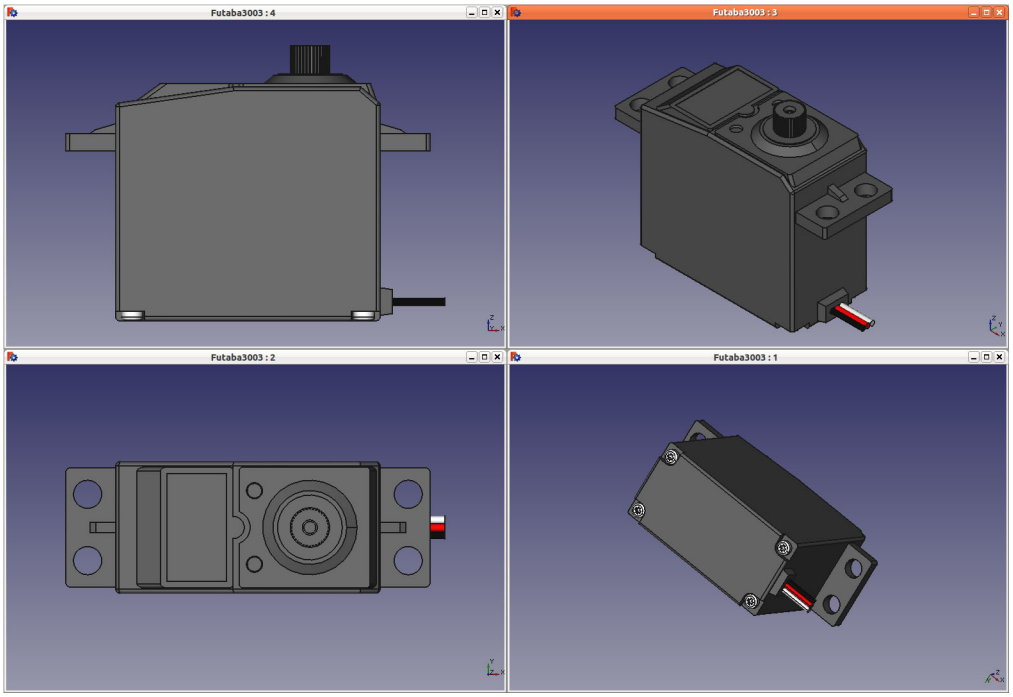


Figura 3.5: *Diseño CAD de los servos del robot [57]*

3.2.2. Ensamblaje del modelo

Una vez diseñadas todas las piezas que componen el modelo, se procede a utilizar dichos diseños CAD para realizar un montaje del modelo en V-REP, obteniendo de esta manera una escena que contiene un modelo del robot con el que poder simular. Dicho modelo, gracias a las funcionalidades de V-REP, puede exportarse y ser utilizado en otras escenas si se desea.

Para comenzar, es necesario convertir primero los archivos a formato ".stl", que es el que puede ser importado en V-REP. Una vez convertidos a ".stl", se sigue la siguiente ruta en el menú [File –>Import –>Mesh...] y se selecionan los distintos archivos para importar cada pieza en V-REP. Al hacer esto, V-REP te da la opción de escalar las piezas que se importan para su correcto montaje, como se puede apreciar en la imagen izquierda de la Figura 3.6. Mediante este procedimiento se importan una a una todas las piezas que componen el modelo.

En la jerarquía del modelo se puede, además, renombrar los objetos haciendo doble click en el nombre actual. Se procede a nombrar, por comodidad y organización, los codos como *Link1_i* y los pies como *Link2_i*, donde *i* va de 1 hasta 4 (debido a las 4 patas existentes). De tal manera que, por ejemplo, *Link1_3* significa "Link 1 (codo) de la pata 3" y, por ejemplo, *Link2_1* significa "Link 2 (pie) de la pata 1". Cada uno de los servos se renombra a Servo*ij*, donde *i* es 1 si el servo está unido a un codo y 2 si está unido a un pie y *j* va de 1 hasta 4 (debido a las 4 patas existentes). De tal manera que, por ejemplo, *Servo14* significa "Servo 1 (ligado al codo) de la pata 4" y, por ejemplo, *Servo21* significa "Servo 2 (ligado al pie) de la pata 1". El objeto que representa el cuerpo del robot se renombra a *Body*. La jerarquía resultante de este proceso se muestra en la imagen derecha de la Figura 3.6.

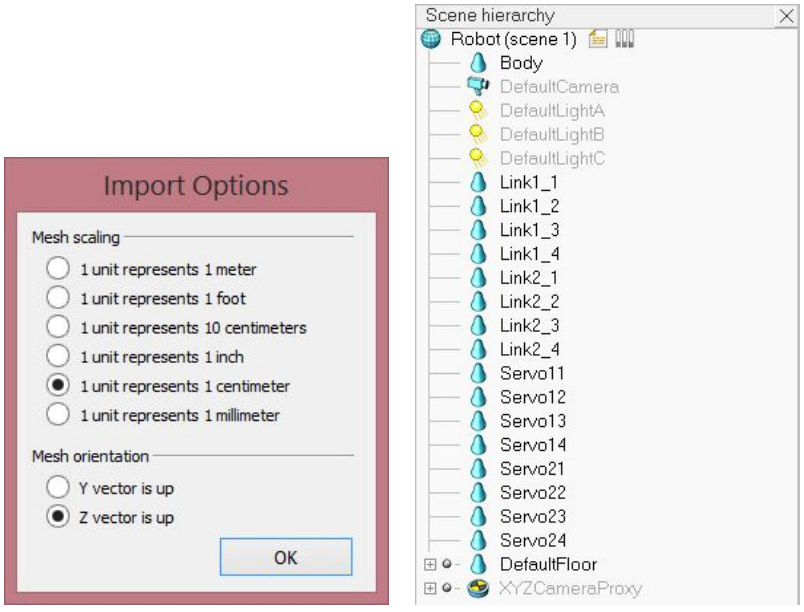


Figura 3.6: *Ventana de escalado y Nombres de objetos*

Debe tenerse en cuenta que el color de las piezas importadas en V-REP por defecto es aleatorio. Para poder definir el color que se desea en cada pieza, se hace doble click en el icono de la pieza en la jerarquía situada en la parte izquierda de la escena de simulación, como se muestra en la imagen izquierda de la Figura 3.7. A continuación aparece un menú de propiedades de objeto como el mostrado en la imagen derecha de la Figura 3.7.

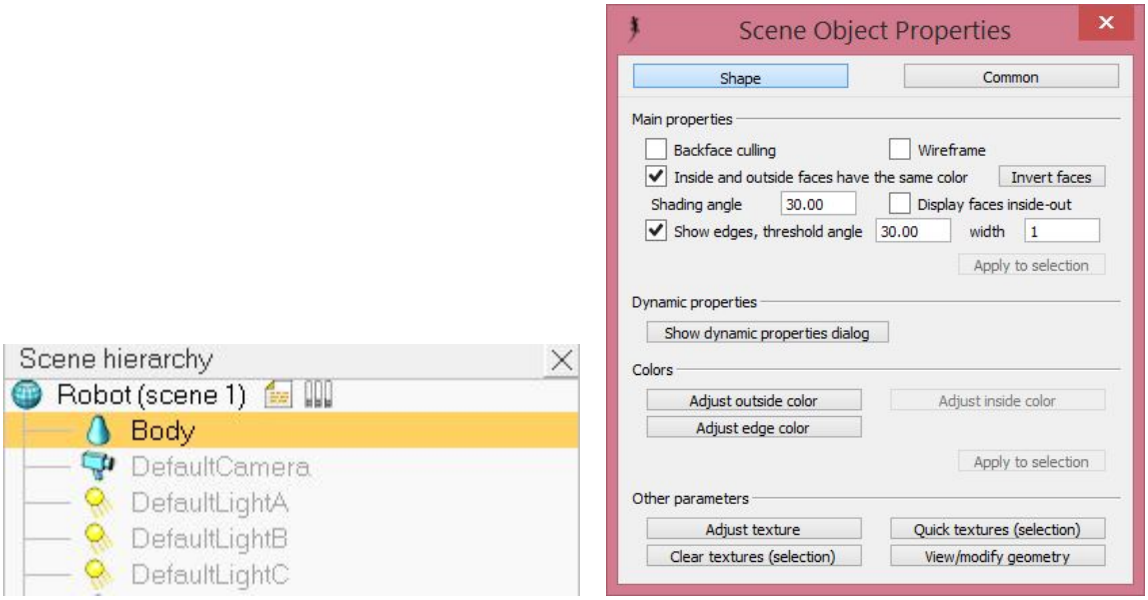


Figura 3.7: *Jerarquía de la escena y Menú de propiedades de objeto*

En dicho menú de propiedades de objeto, siguiendo la siguiente ruta [Shape –>Adjust outside color], se accede al menú de color mostrado en la imagen izquierda de la Figura 3.8. Haciendo click en [Ambient/diffuse component] se accede a la ventana mostrada en la imagen derecha de la Figura 3.8 donde se puede ajustar el código RGB del color deseado.

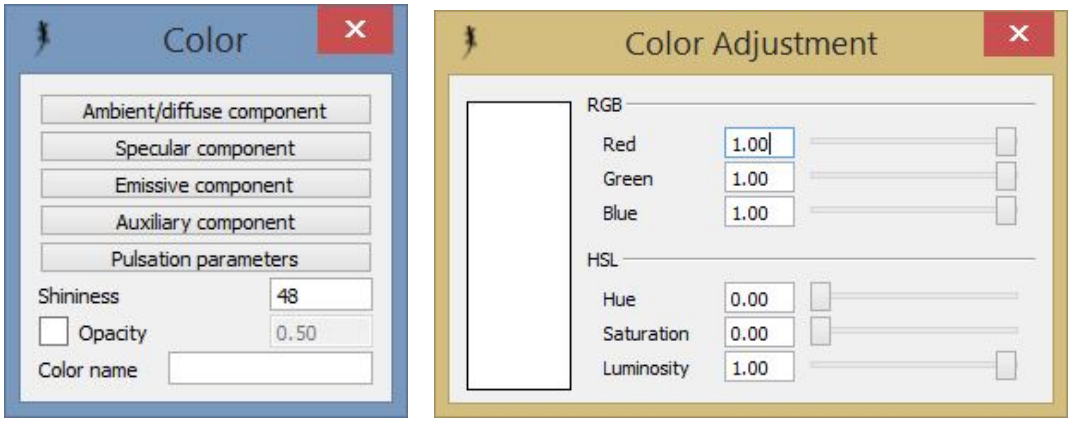


Figura 3.8: *Menú de color y Menú de ajuste RGB*

Una vez todas las piezas están importadas en la escena y tienen los colores deseados, teniendo en cuenta las medidas del robot, se pueden posicionar y orientar para que encajen de manera correcta. Para acceder a los menús correspondientes para posicionar y orientar las piezas, lo primero es hacer un solo click en el objeto deseado en la jerarquía de la escena. Después, se debe pinchar en los botones mostrados en la Figura 3.9. De esta forma se accede al menú de posición y de orientación, respectivamente, y se pueden entonces definir posiciones y orientaciones de la pieza. Ambos menús se muestran, asimismo, en la Figura 3.10.

Figura 3.9: *Botones de acceso a los menús de posición y orientación*

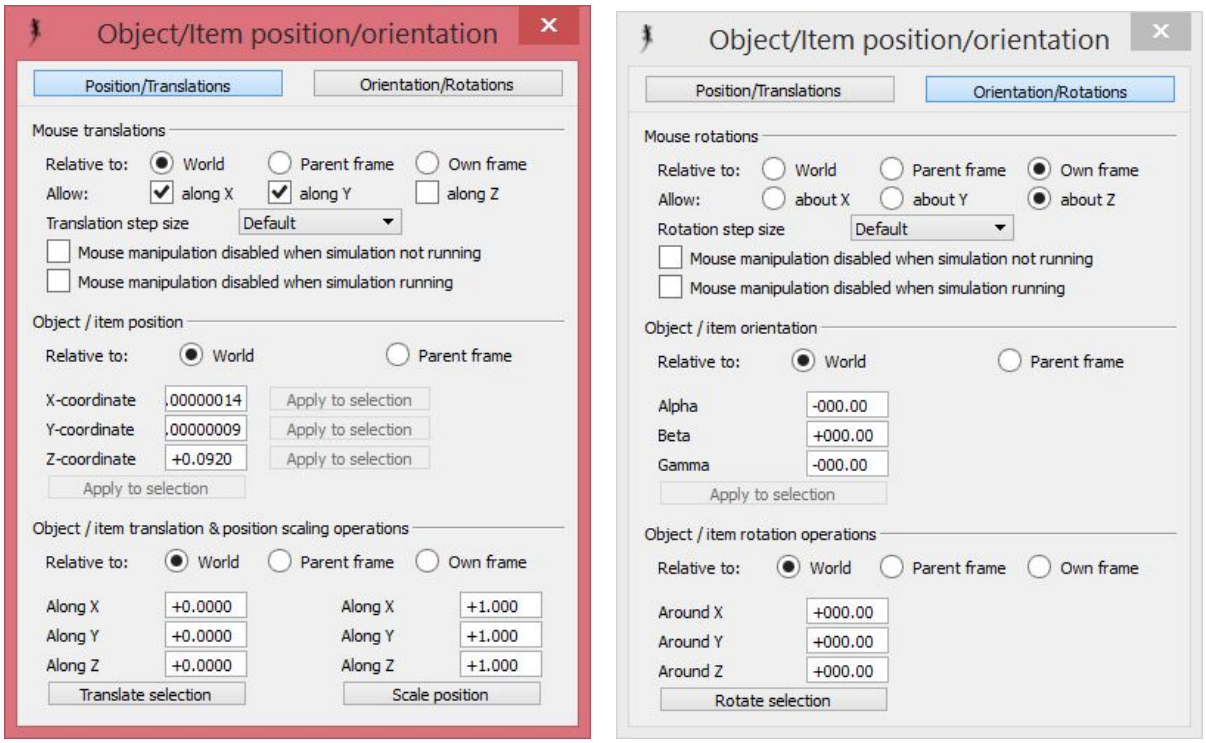


Figura 3.10: *Menú de posición y Menú de orientación*

Llegados a este punto, se tienen todas las piezas en el lugar correcto y con la orientación adecuada. Además, las piezas tienen el mismo color que en el robot real. La escena de simulación que resulta es la mostrada en la Figura 3.11.

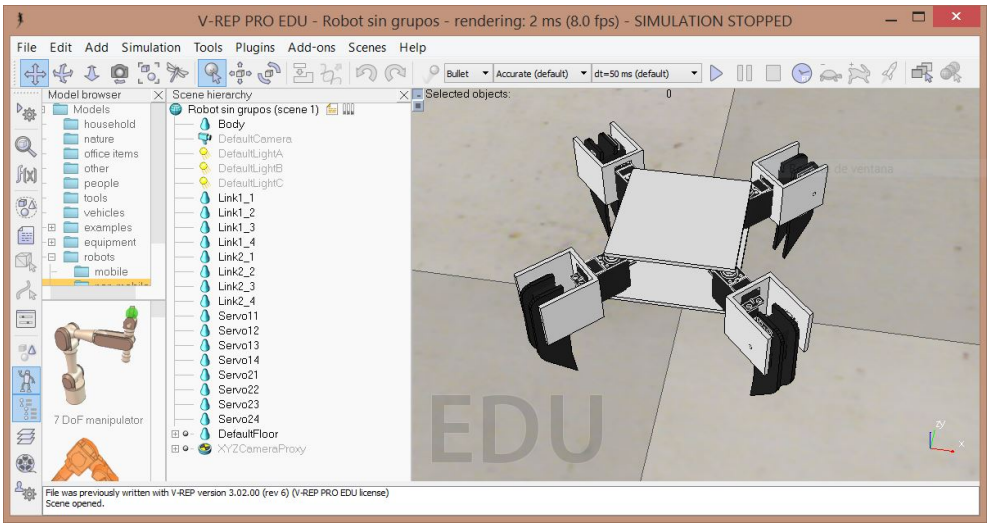


Figura 3.11: *Modelo del robot posicionado, orientado y coloreado*

En el robot real, cada servo es solidario a una parte de la pata del robot. De tal forma que el *Servo21*, por ejemplo, se encuentra ligado al *Link2_1* y es el encargado de mover dicha pieza del robot. Por este motivo, el siguiente paso es agrupar de manera ordenada cada servo con su pieza correspondiente. Para agrupar dos elementos de la jerarquía de la escena se selecciona el *Servo*, se pulsa la tecla "Ctrl" y mientras se mantiene pulsado se selecciona el *Link* que le corresponde. Una vez están los dos objetos seleccionados se sigue la siguiente ruta [Menu bar –>Edit –>Grouping/Merging –>Group selected shapes] y ambos elementos quedan agrupados. Una vez se realiza esta operación con las 8 parejas de servo y link, el resultado es el que se muestra en la Figura 3.12.

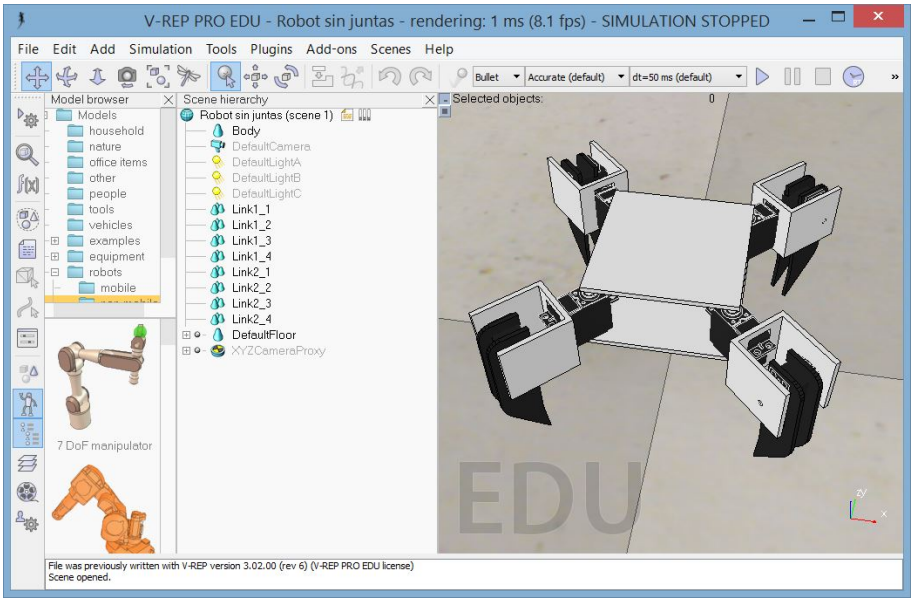


Figura 3.12: *Modelo del robot con grupos*

Con el objetivo de mandar órdenes al modelo de V-REP para que mueva los motores, es necesario añadir elementos que representen ese movimiento. Para ello V-REP dispone de unos objetos llamados *Joints*, que en español podría traducirse como Junta o Articulación. Se incluirán, por tanto, un total de 8 juntas, una por cada servo del robot. Para añadirlas se sigue la siguiente ruta [Menu bar –>Add –>Joint –>Revolute] y el resultado es el mostrado en la Figura 3.13. El resultado es una junta de revolución, pero que no se encuentra en su sitio correcto ni tiene el tamaño adecuado.

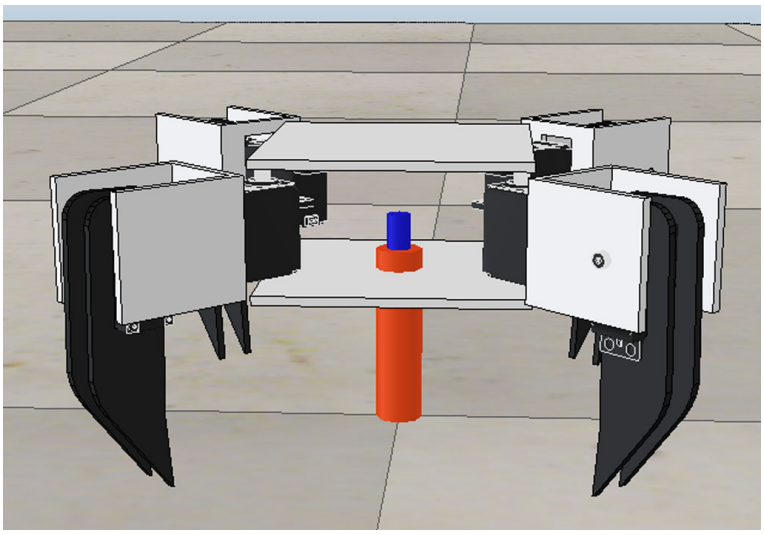


Figura 3.13: *Resultado de añadir una "Revolute joint"*

Para posicionar y orientar la junta como corresponda, se selecciona ésta en la jerarquía de la escena y se utilizan los menús de posición y orientación como se mostró anteriormente. Una vez se encuentra posicionada y orientada se procede a aplicar una escala para que tenga la longitud y el diámetro adecuados. Para ello se accede al menú de propiedades de objeto, como se mostró anteriormente, y se sigue la siguiente ruta [Common –>Scaling]. El menú de escala que aparece es el mostrado en la Figura 3.14.

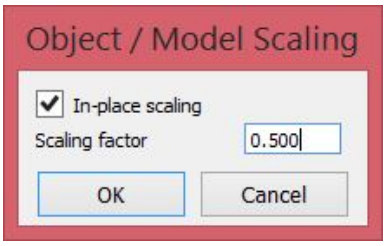


Figura 3.14: *Menú de escala*

Habiendo incluido, posicionado, orientado y escalado las 8 juntas, se procede a renombrarlas como corresponde y el resultado es el mostrado en la Figura 3.15.

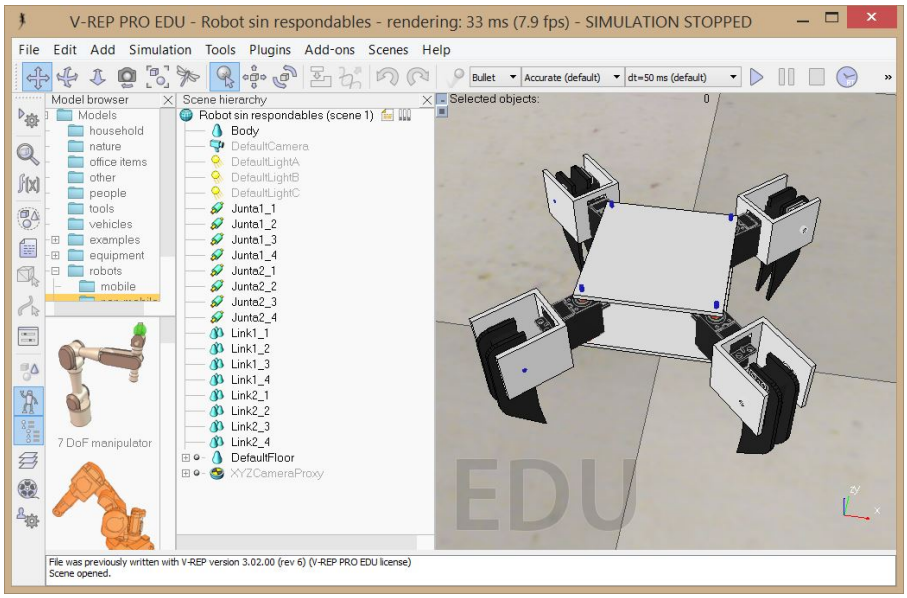


Figura 3.15: *Modelo del robot articulado*

El siguiente paso es preparar los objetos para las simulaciones dinámicas. Para ello existe un tutorial en la web de V-REP bastante extenso, que es el que se ha utilizado [58]. Las masas y momentos de inercia se han configurado para que la simulación sea lo más real posible.

En caso de que durante la simulación se produjera un choque entre dos partes del robot, como pueda ser entre dos patas o entre una pata y el cuerpo, se puede hacer uso del módulo de colisiones de V-REP. Este módulo permite investigar si se producen colisiones entre objetos de la escena y para usarlo se ha seguido el tutorial disponible en la web [59]. En el caso de producirse una colisión, se puede parar en el código el movimiento de un motor para evitar daños o errores. Se han tenido en cuenta las 8 posibles colisiones enumeradas en la siguiente lista y en la Figura 3.16 se muestra la ventana de V-REP con el módulo de colisiones creado.

- *Pata1ConBody*: Colisión entre la pata 1 y el cuerpo.
- *Pata2ConBody*: Colisión entre la pata 2 y el cuerpo.
- *Pata3ConBody*: Colisión entre la pata 3 y el cuerpo.
- *Pata4ConBody*: Colisión entre la pata 4 y el cuerpo.
- *Pata1ConPata2*: Colisión entre la pata 1 y la pata 2.
- *Pata2ConPata3*: Colisión entre la pata 2 y la pata 3.
- *Pata3ConPata4*: Colisión entre la pata 3 y la pata 4.
- *Pata4ConPata1*: Colisión entre la pata 4 y la pata 1.

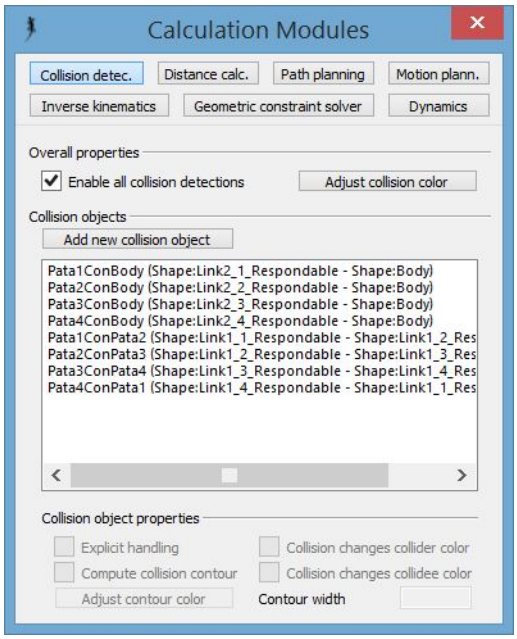


Figura 3.16: *Módulo de colisiones*

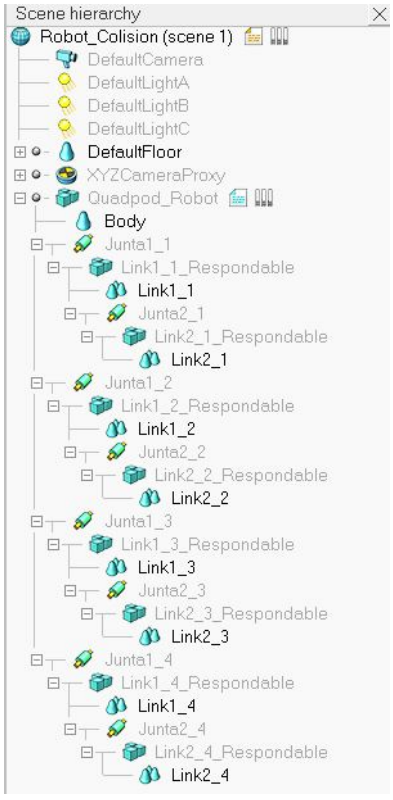


Figura 3.17: *Jerarquía final del modelo del robot*

El último paso consiste en marcar la jerarquía de los objetos dentro del modelo del robot. Para ello se seleccionan objetos de dos en dos y se utiliza la siguiente ruta [Menu bar –>Edit –>Make last selected object parent] para declarar que el último objeto seleccionado de los dos será al que esté encadenado el primer objeto selecionado. Se realizan todos los encadenamientos necesarios hasta obtener una jerarquía como la mostrada en la Figura 3.17, finalizando así el ensamblaje del modelo del robot.

3.3. Entorno de programación. MATLAB

En cuanto a la programación del algoritmo genético, se ha realizado utilizando **Matlab** [60]. Matlab, desarrollado por **MathWorks®**, es un famoso software matemático con un lenguaje propio de alto nivel y con un entorno IDE de desarrollo integrado, del inglés *integrated development environment*. Entre las posibilidades que ofrece Matlab se encuentran el cálculo numérico, el análisis y visualización de datos, la programación de algoritmos, etc.

La elección de Matlab para la programación del algoritmo genético se debe a diversos motivos:

- Es un software realmente veloz en los cálculos y bastante potente como para desarrollar un algoritmo genético, con todos los cálculos que ello conlleva, de manera cómoda.
- Resulta cómodo de programar, debido a su sencillo lenguaje y su integración entre funciones.
- Permite correr el algoritmo varias veces y almacenar los datos para su posterior estudio.
- Es un software con una gran comunidad de usuarios y por tanto, una gran cantidad de información y ayuda disponible.
- El autor ha utilizado dicho programa en anteriores ocasiones y por tanto, no comienza desde cero.

Además, como se comenta en el Apartado 3.1, existe una extensa API con gran cantidad de funciones para manejar, desde el código en Matlab, la simulación de V-REP. Por lo tanto, resulta cómodo poder mandar la instrucción de correr el algoritmo genético en Matlab y dicho software se conectará automáticamente con V-REP para enviar las órdenes para mover la simulación. Tras haber acabado el algoritmo genético, Matlab guardará los resultados obtenidos a los que posteriormente se puede acceder para proceder a su análisis.

En la Figura 3.18 se muestra una ventana de Matlab.

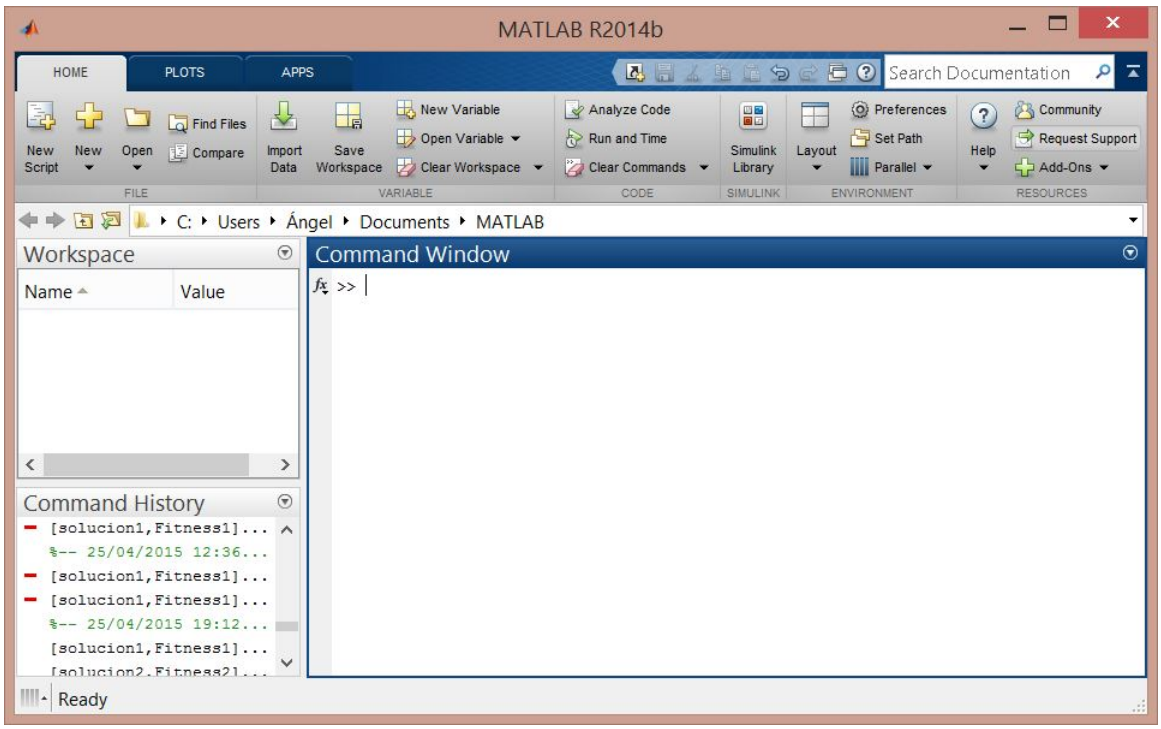


Figura 3.18: *Entorno de programación, Matlab*

3.4. Comunicación entre V-REP y MATLAB

Para poder utilizar la API remota es necesario habilitar su uso tanto desde el servidor (V-REP) como desde el cliente (Matlab). En el Apartado 3.4.1 se aborda la manera de habilitar la API en el lado del servidor y en el Apartado 3.4.2 se aborda la manera de habilitar la API en el lado del cliente.

3.4.1. Habilitar la API en V-REP

La versión de V-REP utilizada es la 3.2.0. Para habilitar la API es necesario iniciar un servidor en V-REP al que posteriormente se conectará el cliente (en este caso matlab). Es posible utilizar, para ello, dos métodos diferentes:

1. **Servidor contínuo**: El servidor será iniciado directamente en el arranque de V-REP y será un servicio contínuo. Para ello es necesario modificar el archivo de configuración del

arranque del programa, con las posibles consecuencias en el correcto funcionamiento que se pueden producir si no se realiza una correcta configuración del arranque.

2. **Servidor temporal**: El servidor será iniciado con una simple instrucción desde un *script* en V-REP al iniciar una simulación y será un servicio temporal que se cerrará al parar la simulación. Un *script* es el lugar donde se indica el código que controla una simulación. De este modo no es necesario modificar ningún archivo de configuración del programa, como en el primer método, y se puede iniciar el servidor cuando se desee.

El método para habilitar la API en V-REP que se ha utilizado es el 2, debido a que es el recomendado por el propio equipo de *Coppelia Robotics*. Lo primero que hay que hacer es crear un *script* y asociarlo al modelo del robot creado. Para ello se pulsa el botón de "Scripts" (mostrado en la imagen izquierda de la Figura 3.19) y se abrirá un menú de scripts como el mostrado en la imagen derecha de la Figura 3.19.

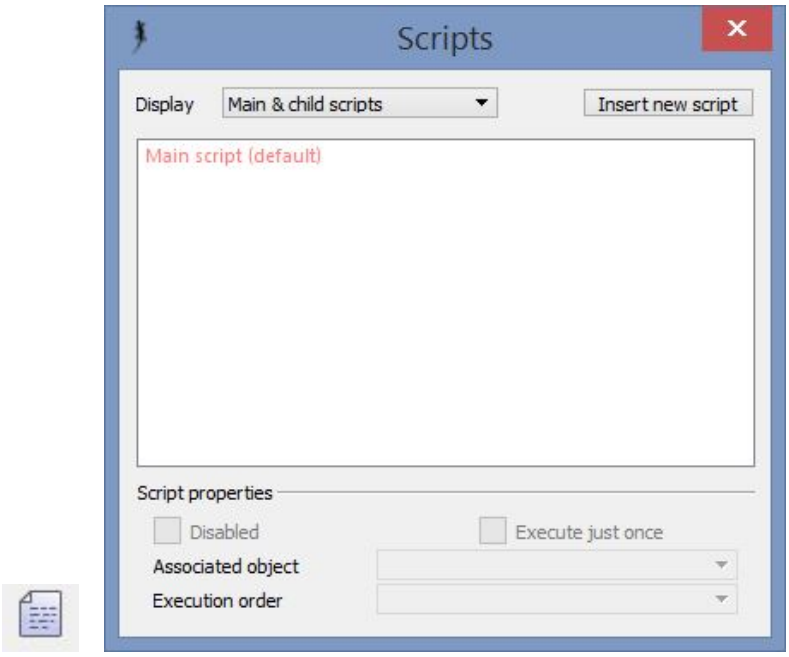


Figura 3.19: *Botón y Menú de scripts*

A continuación, dentro del menú de scripts, se sigue la siguiente ruta [Insert new script –>Child script (threaded) –>OK]. Después se pulsa sobre el script que se ha generado y se asocia al modelo del robot llamado "Quadpod_Robot" mediante las opciones de la parte inferior del menú. El resultado se muestra en la Figura 3.20.

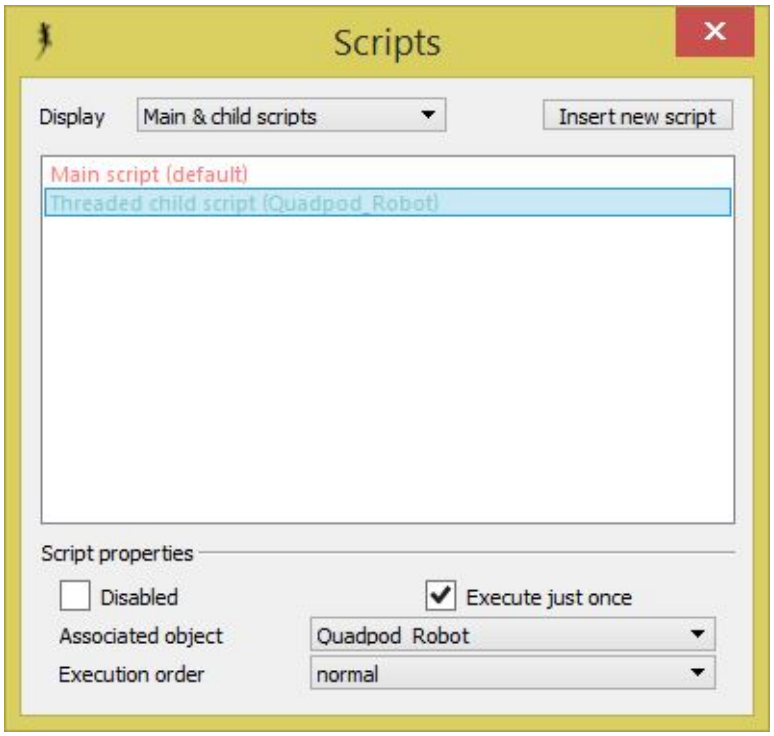


Figura 3.20: *Script asociado al robot*

Una vez hecho esto, se habrá asociado el script al modelo del robot creado y únicamente quedará escribir en él la instrucción que inicia el servidor de la simulación al que nos conectaremos desde Matlab posteriormente. Para ello se hace doble click sobre el icono del script que ha aparecido junto al nombre del modelo del robot en la jerarquía de la escena y se escribe la siguiente instrucción (en la que 19999 es el puerto en el que se creará el servidor y al cual habrá que conectarse desde Matlab):

```
simExtRemoteApiStart(19999)
```

De esta manera la API remota queda habilitada en la parte de V-REP. En la referencia bibliográfica [61] se ofrecen detalles acerca del método interno de funcionamiento de la API remota. Se explica la manera que tiene el servidor de V-REP de recibir las órdenes, almacenarlas, ejecutarlas y devolver los resultados que se deseean y la manera en que dichas instrucciones y resultados salen y entran al cliente que se encuentre utilizando el servidor remoto.

3.4.2. Habilitar la API en Matlab

La versión de Matlab utilizada es la "R2014b". Para poder utilizar las funcionalidades de la API remota en el programa Matlab es necesario copiar, en la carpeta donde se tienen los archivos que contienen el código de matlab, los siguientes archivos:

- remoteApiProto.m
- remApi.m
- remoteApi.dll

Los archivos citados se encuentran alojados en el directorio de instalación de V-REP, bajo la siguiente ruta [programming/remoteApiBindings/matlab]. Al utilizar el segundo método de creación del servidor en el lado de V-REP (ver Apartado 3.4.1), es necesario iniciar la simulación con el botón de *play* desde V-REP antes de ejecutar el código del programa en Matlab.

Una vez los archivos han sido copiados a la carpeta de trabajo de Matlab y se ha iniciado la simulación, se puede ejecutar código desde matlab y mandar órdenes a V-REP. Para ello es necesario realizar una llamada, *vrep=remApi('remoteApi')*, al inicio del código para crear el objeto servidor y cargar la librería de funciones. Tras esto, es necesario realizar otra llamada, *vrep.simxStart*, para habilitar el acceso a dicho servidor (accediendo así al puerto 19999 creado en el script de V-REP, como se explica en el Apartado 3.4.1). Es recomendable hacer una llamada intermedia entre las dos anteriores para cerrar todas las conexiones previamente abiertas, por si acaso. Asimismo se cierra la conexión al final del código mediante el uso de las instrucciones *vrep.simxFinish* y *vrep.delete()*. A continuación se muestra un ejemplo de código para establecer conexión entre Matlab y V-REP:

Ejemplo de programa de conexión:

```
vrep=remApi('remoteApi'); %Se crea el archivo servidor (utiliza
    remoteApiProto.m)
vrep.simxFinish(-1); %Por si acaso, se cierran todas las conexiones
    previamente abiertas
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5); %Se establece
    el cliente de vrep
if (clientID>-1)
        %Introducir aqui el codigo de lo que se quiera hacer%
        vrep.simxFinish(clientID); %cerramos la conexion
else
        %En caso de entrar aqui significa que se ha producido un error en
            la conexion%
end;
vrep.delete(); %Se llama al destructor
```

En la referencia bibliográfica [62] se recoge un listado con todas las funciones que contiene la API remota entre Matlab y V-REP. Nótese que es necesario escribir delante de las funciones el prefijo "vrep.simx". Se utiliza la misma arquitectura (64 bits) tanto en Matlab como en la librería *remoteApi*. Esto se debe a que un Matlab de 64 bits no funciona con una librería *remoteApi* de 32 bits, y viceversa.

3.5. Pruebas del entorno de trabajo

Han sido realizadas varias pruebas con los siguientes objetivos:

1. Comprobar que se establecen las conexiones de manera satisfactoria.
2. Comprobar la validez del modelo del robot real creado de la manera descrita en el Apartado 3.2.2.
3. Lograr programar y hacer avanzar el modelo de la simulación con un patrón de avance prediseñado.
4. Lograr medir las posiciones inicial y final del robot para poder así calcular la distancia avanzada, con el objetivo de implementar dicha distancia como función de *fitness* en el algoritmo genético.

Para dichas pruebas se ha utilizado el código que se encuentra en el Anexo II. Se comprueba que la conexión entre Matlab y V-REP se establece correctamente, el robot consigue avanzar gracias al patrón de avance prediseñado y al modelo realizado en V-REP y a su vez se logra medir las posiciones del robot en la simulación y obtener así el avance en Matlab. Debido al éxito de dichas pruebas, puesto que se cumplen los objetivos fijados, se valida el entorno de trabajo y se pasa a programar el algoritmo genético.

4

Diseño del Algoritmo Genético

En este capítulo se va a abordar el algoritmo genético como tal que se ha desarrollao. Para comenzar se describe el tipo de individuos por el que se ha optado, en el Apartado 4.1, y posteriormente se expone la estructura con la que se ha diseñado el algoritmo, en el Apartado 4.2.

A modo de resumen de lo expuesto en el Apartado 2.5 y a modo de introducción al presente capítulo, cabe destacar que un algoritmo genético se compone de una población de individuos, inicialmente aleatorios, codificados de una manera similar a cromosomas. Cada individuo tendrá asociado un valor de *fitness*, que define cómo de bien soluciona ese individuo el problema. Los individuos se irán cruzando en cada generación y la población irá avanzando, en principio, hacia soluciones mejores. Además, podrán producirse mutaciones con una cierta probabilidad.

4.1. Codificación de los individuos

La codificación de los individuos es una cuestión importante y complicada a la hora de diseñar un algoritmo genético. Distintas codificaciones de individuos pueden conducir a distintos resultados durante la evolución del algoritmo. Además, en relación a la codificación del individuo, se podrán utilizar diferentes tipos de cruce y mutación. Se procede, por tanto, a proponer varios tipos de codificación de los individuos y a elegir una de ellas. Las codificaciones que se han barajado son las siguientes:

- Una codificación binaria que se componga por una secuencia de valores de posiciones de ángulos de los servos. El individuo estaría compuesto por todos los bits de las distintas posiciones que iría tomando el robot. Dicho de otra manera, cada individuo representa una corta secuencia de marcha.
- Una codificación no binaria en la que cada motor se mueva de una manera senoidal, hacia delante y luego hacia atrás contínuamente. El individuo se compondría, por tanto, de dos parámetros para cada motor: amplitud y frecuencia de la senoide.
- Otras codificaciones no binarias similares a la anterior, pero con señales triangulares o cuadradas. En estos casos los parámetros para cada motor que compondrían el individuo podrían ser la amplitud de la señal y su período.

Tras un análisis de las posibilidades que brinda cada una de las opciones, se ha optado por elegir el **segundo tipo** de codificación. Entre los motivos de esta elección está el hecho de que existen bastantes publicaciones que se centran en codificaciones de tipo binario y muy pocas que utilizan codificaciones no binarias (con números reales o enteros). Además de un reto, este tipo de codificación supone, por tanto, una innovación frente a lo que se utiliza normalmente.

La codificación del cromosoma de los individuos es, por tanto, no binaria. En cuanto a la idea original de definir cada senoide mediante dos parámetros $A \equiv amplitud$ y $\omega \equiv frecuencia$, se introduce un tercer parámetro que es el $\varphi \equiv retraso$ de la senoide. Teniendo un parámetro más, el abanico de posibles soluciones aumenta y el problema de avanzar queda mejor definido.

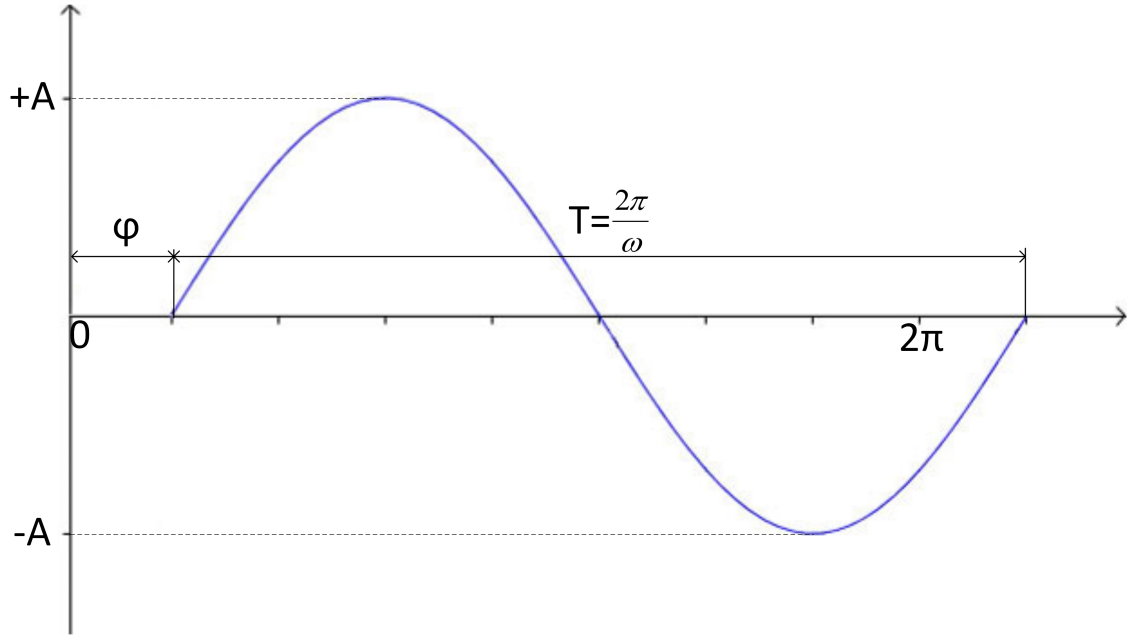


Figura 4.1: *Seno definido por los parámetros*

El esquema de la codificación del cromosoma de un individuo se muestra en la Figura 4.2, donde se suceden los 3 parámetros anteriormente mentados para cada uno de los 8 motores que tiene el robot. De esta forma el individuo queda definido como un conjunto de $8 \times 3 = 24$ parámetros, siendo todos ellos números reales.

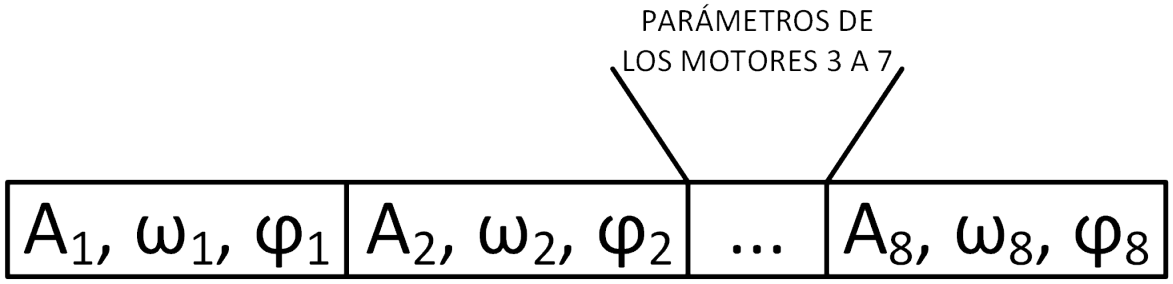


Figura 4.2: *Codificación de los individuos*

Se deben definir intervalos entre los que se encuentren cada uno de los tres parámetros. Cuando la población inicial sea creada, cada parámetro, para cada motor, tomará un valor aleatorio comprendido entre los límites de su intervalo. Con el objetivo de conseguir unos resultados óptimos, los intervalos entre los que se encuentran los tres parámetros que definen la senoide con las que se mueve cada motor son:

- $A \in [0,45]$
- $\omega \in [\frac{\pi}{3}, \pi]$
- $\varphi \in [0, 2\pi]$

Evidentemente, la orden para el motor no puede ser mandar directamente el seno, porque el motor no entiende que se está moviendo como un seno, si no que se le debe mandar un ángulo de posición al que desplazarse. Para ello será necesario muestrear cada seno con una serie de valores que deberán ser mandados a cada motor. Se ha utilizado un valor de muestreo de N=5 valores para cada seno. De este modo, para cada valor muestreado Vi, con i tomando valores desde 1 hasta N, las ecuaciones de muestreo son la 4.1 y la 4.2.

$$t = i \cdot \frac{2\pi}{N} \tag{4.1}$$

$$Vi = A \cdot sin(\omega t + \varphi) \tag{4.2}$$

Los ángulos de posición que van tomando los motores, al muestrear el seno, son tanto positivos como negativos. Esto quiere decir que el motor se mueve hacia un lado y luego hacia el contrario de una manera contínua. En la Figura 4.3 se muestra un esquema de la dirección que toman los motores M1, M3, M5 y M7 en función del signo del ángulo de posición. Dicho esquema se muestra en planta. En la Figura 4.4 se muestra un esquema de la dirección que toma cualquiera de los motores M2, M4, M6 y M8 en función del signo del ángulo de posición. Dicho esquema se muestra en perfil.

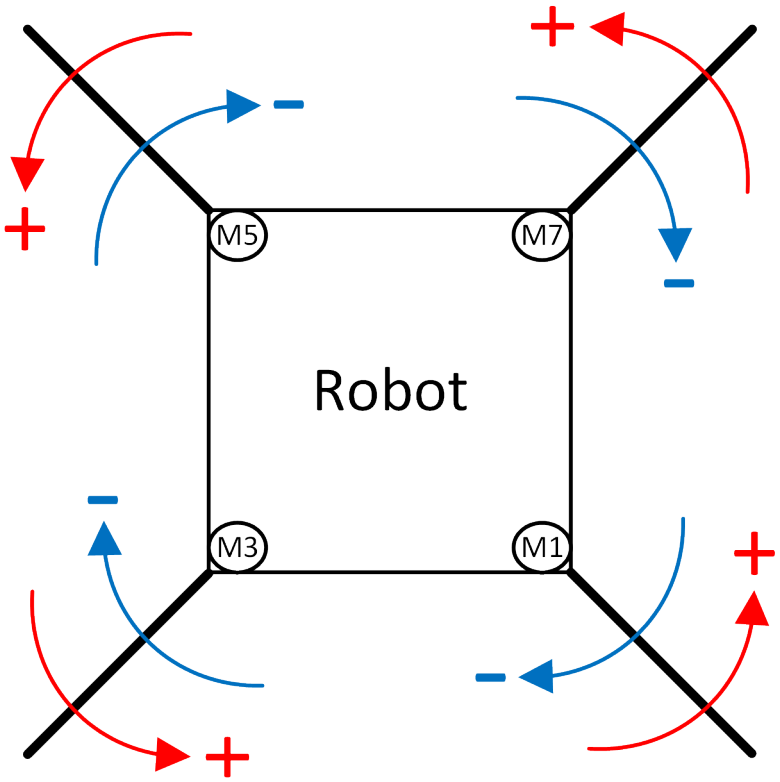


Figura 4.3: *Esquema de dirección de motores M1, M3, M5 y M7*

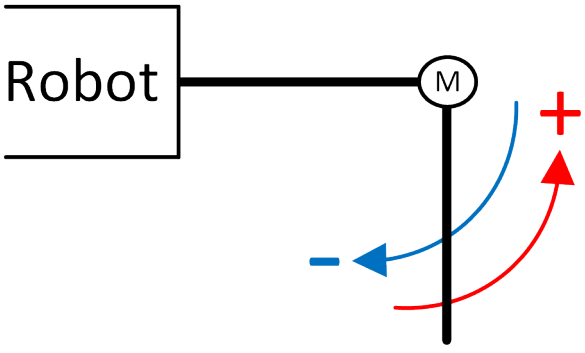


Figura 4.4: *Esquema de dirección de motores M2, M4, M6 y M8*

Por último, en la Tabla 4.1 se muestra una relación entre el número de los motores y los nombres en el modelo de V-REP. Los nombres del modelo en V-REP son los que se explican en el Apartado 3.2.2.

Número del motor	Nombre en V-REP
M1	Junta1_1
M2	Junta2_1
M3	Junta1_2
M4	Junta2_2
M5	Junta1_3
M6	Junta2_3
M7	Junta1_4
M8	Junta2_4

Tabla 4.1: Correspondencia entre el número de cada motor y su nombre en el modelo de V-REP creado

4.2. Estructura del algoritmo

En este apartado se aborda el algoritmo genético que se ha desarrollado. En el Anexo III ha sido incluido, para su consulta, el código del algoritmo al completo. El algoritmo ha sido programado en Matlab, como ya se ha comentado, y ha sido probado en V-REP. Se ha dividido en las siguientes 8 funciones para una mejor estructuración y depuración del código:

- **genetico**: Calcula la solución óptima para maximizar el avance del robot mediante algoritmos genéticos. Se encarga de gestionar el algoritmo, haciendo llamadas a funciones, y devolver la solución cuando se termine. Vigila el criterio de finalización, que puede ser por el fin de las generaciones requeridas o por la convergencia de la población. Además, al terminar, hace moverse al robot durante unos cuantos pasos con la solución obtenida. Esta función tiene 5 argumentos de entrada utilizados para definir la manera deseada de funcionamiento del algoritmo. Por orden, son los siguientes:
  - **NumGeneraciones**: Indica el número de generaciones que se van a suceder.
  - **NumIndividuos**: Indica el número de individuos de la población.
  - **LimMejora**: Se utiliza para controlar si el algoritmo converge.
  - **pC**: Tasa de cruce deseada.
  - **pM**: Tasa de mutación deseada.
- **PoblacionInicial**: Su misión es crear la población inicial de individuos aleatorios. En el Apartado 4.1 se explica el tipo de codificación utilizada. Devuelve una matriz de tantas filas como individuos y tantas columnas como parámetros.

- **Cruce**: Se encarga de realizar los cruces de los individuos con una probabilidad *Pc* y, si es necesario, realiza las mutaciones con una probabilidad *Pm*. En el caso de que no se deba producir el cruce, se encarga de copiar uno de los progenitores. Cada nuevo individuo que surge se añade a la población sin eliminar a ningún otro ya existente, y así hasta alcanzar el doble de individuos que componen la población. Esta función devuelve, por tanto, una matriz de tantas filas como el doble de individuos y tantas columnas como parámetros.

- **Seleccion**: Utiliza selección por *ranking*. Se encarga de ordenar los individuos en función de su *fitness* (llamando a la función *EvaluarFitness*) y de eliminar a la "peor mitad". Esta función devuelve, de esta manera, una matriz de tantas filas como individuos y tantas columnas como parámetros. Además devuelve el *fitness* del individuo más apto.

- **EvaluarFitness**: Se encarga de probar uno por uno cada individuo de la población que se le pasa y obtener su *fitness* (avance del robot con ese individuo). Para obtener el *fitness* de un individuo se realizan llamadas a las funciones *Traducir* y *Mover*. Esta función devuelve un vector fila en el que cada elemento es el *fitness* de cada individuo, con el objetivo posterior de ordenar en la funcion *Seleccion* y así escoger a la mejor mitad.

- **Traducir**: El objetivo de esta funcción es recibir un individuo codificado mediante parámetros y traducirlo a secuencias de ángulos de posiciones para los motores. Realiza un muestreo del seno definido por los parámetros del individuo y devuelve una matriz donde están muestreados todos los senos de los motores. Dicha matriz tiene tantas filas como motores y tantas columnas como muestras obtenidas. Para traducir se utilizan las ecuaciones 4.1 y 4.2, explicadas en el Apartado 4.1.

- **Mover**: Su finalidad es conectarse con V-REP (ver Apartado 3.4) y mover el robot con un individuo previamente muestreado en la función *Traducir*. Devuelve las posiciones inicial y final del robot, de tal forma que después se pueda obtener el avance del robot en la función *EvaluarFitness*.

- **MoverFinal**: Una vez el algoritmo genético ha finalizado, se utiliza esta función para conectarse con V-REP y mover el robot durante algunos pasos con el objetivo de mostrar visualmente el avance logrado. Se diferencia de la función *Mover* en que no devuelve las posiciones inicial y final del robot, sino que tan solo se encarga de moverlo unas cuantas veces para mejorar el apartado visual.

Con el propósito de comprender mejor el funcionamiento del algoritmo y las llamadas entre funciones, en la Figura 4.5 se muestra un diagrama de flujo. Para mayor detalle del funcionamiento interno de cada función, acudir al Anexo III.

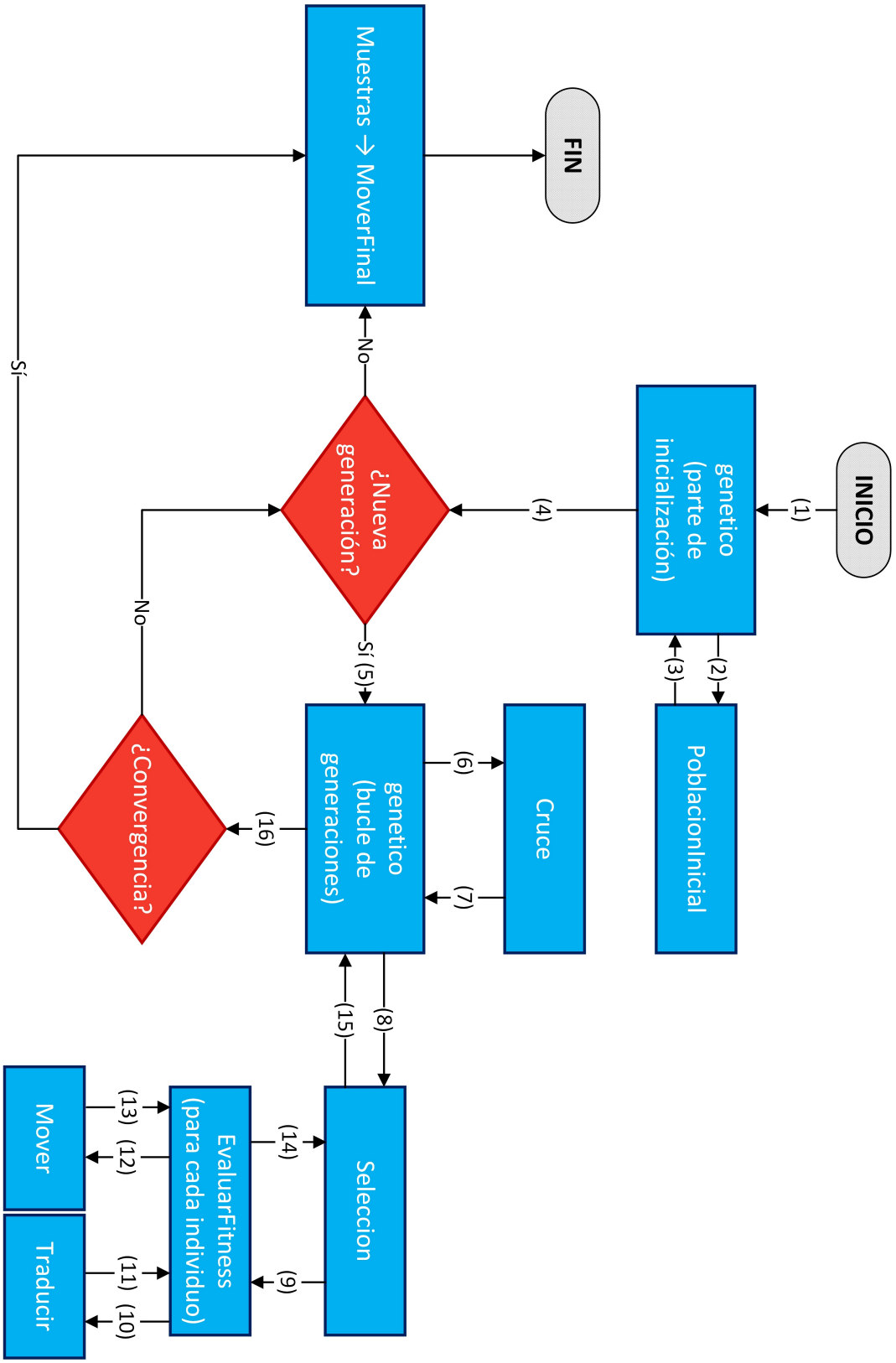


Figura 4.5: *Diagrama de flujo del funcionamiento del Algoritmo Genético desarrollado*

5

Simulación del algoritmo

La realización de pruebas de un algoritmo genético requiere gran cantidad de tiempo, debido a que se debe probar, en cada generación, individuo por individuo en el robot y obtener el *fitness* de cada uno para un correcto funcionamiento. Además, es objeto del presente proyecto realizar un estudio de distintas combinaciones del algoritmo, de cara a obtener los mejores resultados.

En consecuencia a las premisas expuestas, se antoja inviable probar cada combinación del algoritmo directamente en el robot real. Por ello, se ha utilizado el simulador desarrollado (ver Capítulo 3) para hacer un estudio de las distintas combinaciones posibles y de esta manera ahorrar tiempo. A las vistas de los resultados obtenidos en el presente capítulo, en el Capítulo 6 se exponen los experimentos posteriores realizados con el robot real. A su vez, cabe recordar que el *fitness* es la medida de aptitud de un individuo para el problema planteado y se trata de maximizarlo. La medida de bondad utilizada es el **avance** logrado.

5.1. Variando parámetros de control

Los **parámetros de control** del algoritmo que se han utilizado son el número de generaciones (**NumGeneraciones**), el número de individuos (**NumIndividuos**), la tasa de cruce (**pC**) y la tasa de mutación (**pM**). Por ello, en la presente sección, se ha procedido a realizar una serie de simulaciones con objeto de determinar la manera en que el *fitness* varía en función de cada uno de los parámetros de control.

Debido al elevado número de valores posibles, se han realizado simulaciones de manera que se abarquen la mayor parte de estas posibilidades de la mejor manera posible. Además, en consecuencia a la propia naturaleza aleatoria de un algoritmo genético, de cada prueba han sido realizadas varias simulaciones y se han valorado sus medias.

5.1.1. Variando el número de generaciones

Han sido estudiados **6** posibles valores para el **número de generaciones**, con **5** experimentos (repeticiones) en cada caso. De cara a estas simulaciones, se han tomado los siguientes valores para el resto de parámetros de control:

- NumIndividuos = 20
- pC = 0.9
- pM = 0.02

Los resultados obtenidos se muestran en la Tabla 5.1.

NumGeneraciones	Experimento	*Fitness*
4	1	0.0991
4	2	0.0955
4	3	0.1021
4	4	0.1242
4	5	0.0727
4	**Media**	**0.09872**
10	1	0.1224
10	2	0.1357
10	3	0.1041
10	4	0.1131
10	5	0.1356
10	**Media**	**0.12218**
25	1	0.1332
25	2	0.1050
25	3	0.1226
25	4	0.1451
25	5	0.1067
25	**Media**	**0.12252**
50	1	0.1059
50	2	0.1641
50	3	0.1182
50	4	0.1205
50	5	0.1478
50	**Media**	**0.1313**
75	1	0.1414
75	2	0.1257
75	3	0.1284
75	4	0.1192
75	5	0.1449
75	**Media**	**0.13192**
100	1	0.1529
100	2	0.1774
100	3	0.1111
100	4	0.1792
100	5	0.1235
100	**Media**	**0.14882**

Tabla 5.1: Variando el número de generaciones

En la Figura 5.1 se muestra un gráfico donde se representa la evolución del *fitness* según varía el número de generaciones del algoritmo genético.

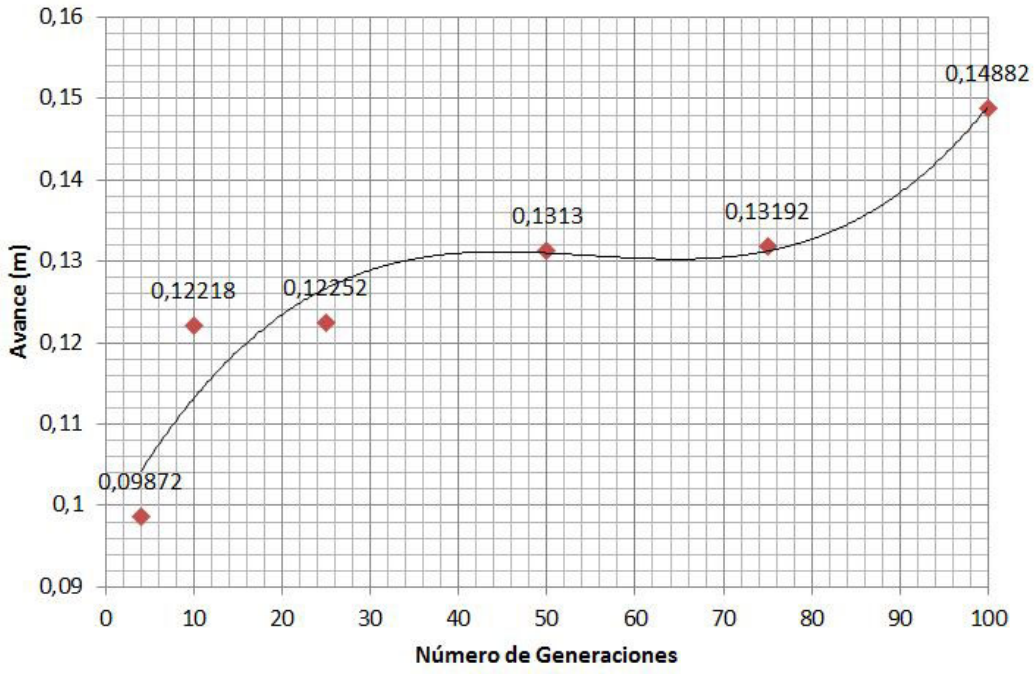


Figura 5.1: *Avance en función del Número de Generaciones*

La línea negra representa una línea de tendencia polinómica de tercer orden que se aproxima a los valores medios obtenidos. Dicha línea tiene por ecuación la 5.1.

$$y = 2 \cdot 10^{-7} x^3 - 4 \cdot 10^{-5} x^2 + 0{,}002x + 0{,}0968 \tag{5.1}$$

5.1.2. Variando el número de individuos

Han sido estudiados **6** posibles valores para el **número de individuos**, con **5** experimentos (repeticiones) en cada caso. De cara a estas simulaciones, se han tomado los siguientes valores para el resto de parámetros de control:

- NumGeneraciones = 20
- pC = 0.9
- pM = 0.02

Los resultados obtenidos se muestran en la Tabla 5.2.

NumIndividuos	**Experimento**	***Fitness***
4	1	0.0878
4	2	0.0885
4	3	0.0596
4	4	0.0658
4	5	0.0683
4	**Media**	**0.074**
10	1	0.0989
10	2	0.1038
10	3	0.1044
10	4	0.1427
10	5	0.0811
10	**Media**	**0.10618**
25	1	0.1650
25	2	0.1050
25	3	0.1420
25	4	0.1293
25	5	0.1448
25	**Media**	**0.13722**
50	1	0.1159
50	2	0.1406
50	3	0.1249
50	4	0.1721
50	5	0.1429
50	**Media**	**0.13928**
75	1	0.1585
75	2	0.1524
75	3	0.1938
75	4	0.2326
75	5	0.1827
75	**Media**	**0.1840**
100	1	0.1805
100	2	0.2156
100	3	0.1927
100	4	0.1652
100	5	0.1991
100	**Media**	**0.19062**

Tabla 5.2: Variando el número de individuos

En la Figura 5.2 se muestra un gráfico donde se representa la evolución del *fitness* según varía el número de individuos del algoritmo genético.

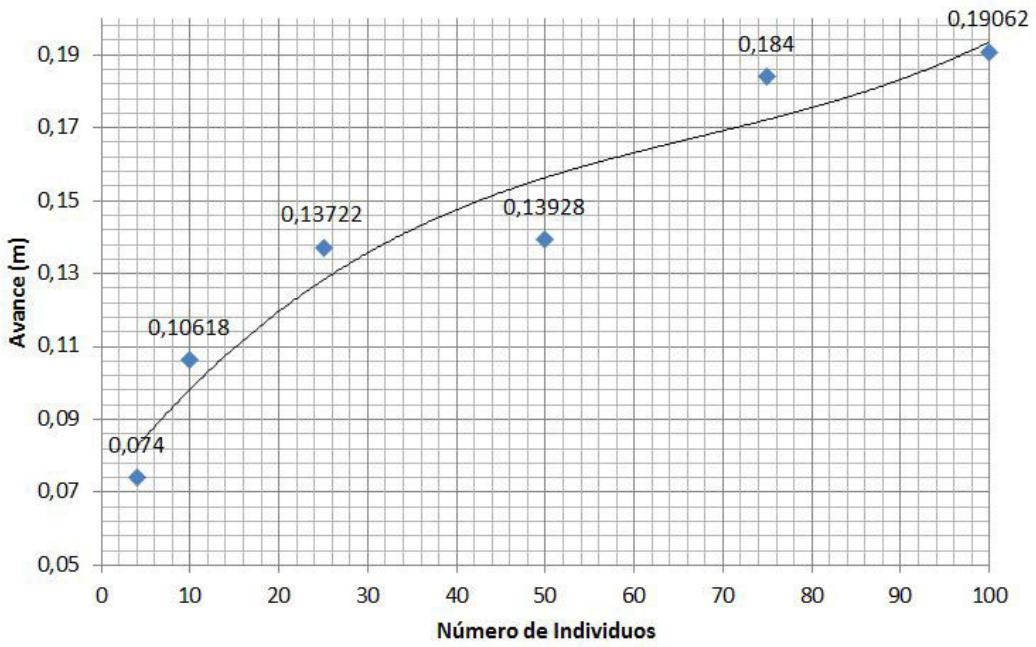


Figura 5.2: *Avance en función del Número de Individuos*

La línea negra representa una línea de tendencia polinómica de tercer orden que se aproxima a los valores medios obtenidos. Dicha línea tiene por ecuación la 5.2.

$$y = 2 \cdot 10^{-7}x^3 - 4 \cdot 10^{-5}x^2 + 0{,}0031x + 0{,}0706 \tag{5.2}$$

5.1.3. Variando la tasa de cruce

Han sido estudiados **6** posibles valores para la **tasa de cruce**, con **5** experimentos (repeticiones) en cada caso. El cruce simulado ha sido un cruce en un punto. De cara a estas simulaciones, se han tomado los siguientes valores para el resto de parámetros de control:

- NumGeneraciones = 20
- NumIndividuos = 20
- pM = 0.02

Los resultados obtenidos se muestran en la Tabla 5.3.

pC	**Experimento**	***Fitness***
0.9	1	0.1264
0.9	2	0.1280
0.9	3	0.1115
0.9	4	0.1461
0.9	5	0.1368
0.9	**Media**	**0.12976**
0.8	1	0.1188
0.8	2	0.1278
0.8	3	0.1272
0.8	4	0.1236
0.8	5	0.1165
0.8	**Media**	**0.12278**
0.75	1	0.1224
0.75	2	0.1448
0.75	3	0.1200
0.75	4	0.1561
0.75	5	0.1081
0.75	**Media**	**0.13028**
0.5	1	0.1661
0.5	2	0.1139
0.5	3	0.1310
0.5	4	0.1046
0.5	5	0.1718
0.5	**Media**	**0.13748**
0.25	1	0.1426
0.25	2	0.0926
0.25	3	0.1059
0.25	4	0.1215
0.25	5	0.1580
0.25	**Media**	**0.12412**
0.1	1	0.0841
0.1	2	0.0969
0.1	3	0.0772
0.1	4	0.0936
0.1	5	0.0863
0.1	**Media**	**0.08762**

Tabla 5.3: Variando la tasa de cruce

En la Figura 5.3 se muestra un gráfico donde se representa la evolución del *fitness* según varía la tasa de cruce del algoritmo genético.

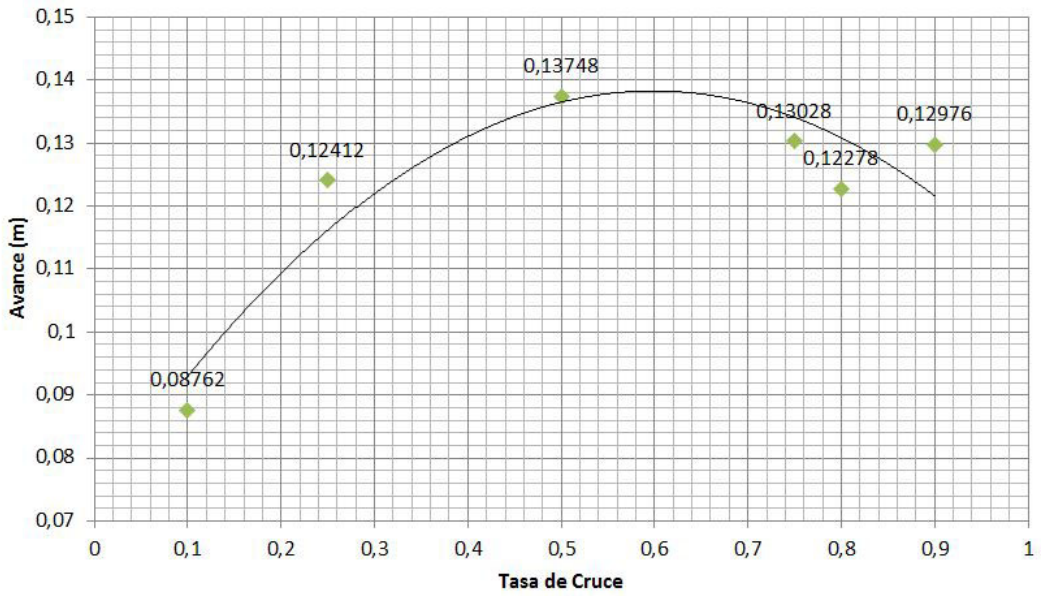


Figura 5.3: *Avance en función de la Tasa de Cruce*

La línea negra representa una línea de tendencia polinómica de segundo orden que se aproxima a los valores medios obtenidos. Dicha línea tiene por ecuación la 5.3.

$$y = -0{,}183x^2 + 0{,}2189x + 0{,}0728 \tag{5.3}$$

5.1.4. Variando la tasa de mutación

Han sido estudiados **6** posibles valores para la **tasa de mutación**, con **5** experimentos (repeticiones) en cada caso. La mutación simulada ha sido una media entre el gen a mutar y un valor aleatorio. De cara a estas simulaciones, se han tomado los siguientes valores para el resto de parámetros de control:

- NumGeneraciones = 20
- NumIndividuos = 20
- pC = 0.9

Los resultados obtenidos se muestran en la Tabla 5.4.

pM	**Experimento**	***Fitness***
0.02	1	0.1367
0.02	2	0.1028
0.02	3	0.1209
0.02	4	0.1549
0.02	5	0.1236
0.02	**Media**	**0.12778**
0.018	1	0.1358
0.018	2	0.1283
0.018	3	0.1243
0.018	4	0.1278
0.018	5	0.1327
0.018	**Media**	**0.12978**
0.016	1	0.1308
0.016	2	0.1311
0.016	3	0.1188
0.016	4	0.1473
0.016	5	0.1265
0.016	**Media**	**0.1309**
0.01	1	0.1474
0.01	2	0.1450
0.01	3	0.1075
0.01	4	0.1236
0.01	5	0.1352
0.01	**Media**	**0.13174**
0.005	1	0.1366
0.005	2	0.1387
0.005	3	0.1364
0.005	4	0.1269
0.005	5	0.1184
0.005	**Media**	**0.1314**
0.002	1	0.1370
0.002	2	0.1308
0.002	3	0.1255
0.002	4	0.1322
0.002	5	0.1321
0.002	**Media**	**0.13152**

Tabla 5.4: Variando la tasa de mutación

En la Figura 5.4 se muestra un gráfico donde se representa la evolución del *fitness* según varía la tasa de mutación del algoritmo genético.

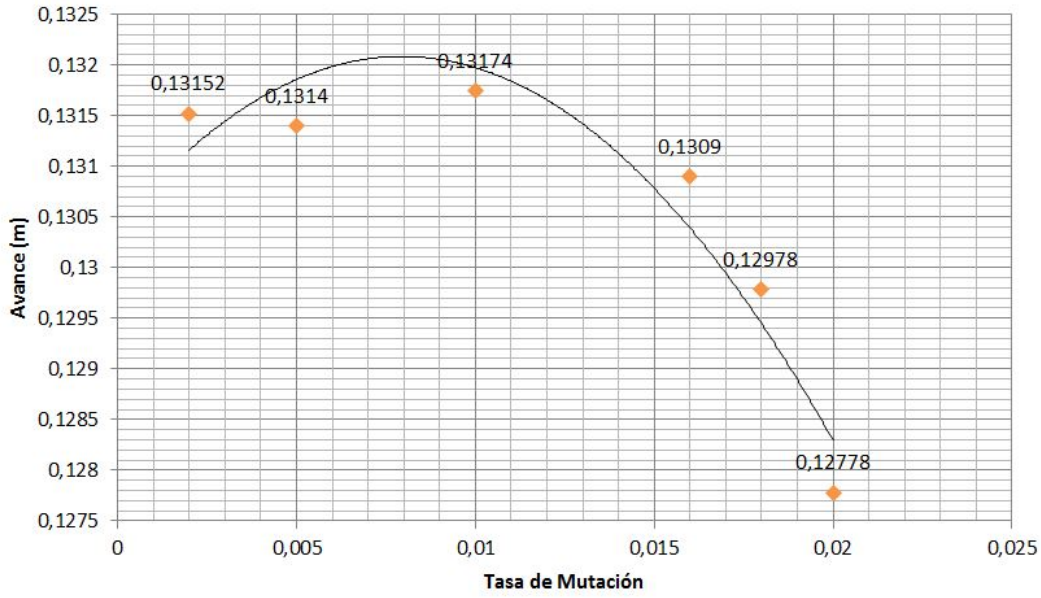


Figura 5.4: *Avance en función de la Tasa de Mutación*

La línea negra representa una línea de tendencia polinómica de segundo orden que se aproxima a los valores medios obtenidos. Dicha línea tiene por ecuación la 5.4.

$$y = -26{,}089x^2 + 0{,}4146x + 0{,}1304 \tag{5.4}$$

5.2. Combinando parámetros de control

Ante el evidente elevado número de combinaciones posibles que surgen de combinar todos los valores de cada uno de los parámetros de control trabajados en el Apartado 5.1, se ha decidido tomar unos cuantos valores de cada parámetro de control y combinarlos entre ellos de cara a conseguir mejores resultados y ampliar el abanico de búsqueda.

Las combinaciones que se han realizado han sido elegidas en relación a los mejores resultados de cada parámetro del Apartado 5.1. A su vez, los valores a combinar han sido escogidos en función al tiempo que conlleva que se complete cada simulación, de manera que el tiempo total invertido en las simulaciones se ajuste al tiempo disponible.

En el presente apartado, de la misma forma que en la primera tanda de simulaciones del apartado anterior, se ha trabajado con cruces en un punto y mutaciones en las que el gen se sustituye por una media entre el propio gen a mutar y un valor aleatorio. Posteriormente en el apartado 5.3, se procede a realizar simulaciones con otros tipos de cruce y de mutación. Dichas simulaciones, como se explica en el citado apartado, se realizan con la mejor combinación de parámetros resultante.

Los valores de los parámetros de control escogidos se listan a continuación:

- **NumGeneraciones**: A la vista de la Figura 5.1, se escogen los valores de **100** y **50** generaciones. El primero de ellos se escoge por ser el que proporciona el mayor avance. Por otra parte, el segundo de ellos proporciona un valor parecido al que se obtiene si se utilizan 75 generaciones, por lo que entre ambos valores se escoge 50 por su menor tiempo de simulación.

- **NumIndividuos**: A la vista de la Figura 5.2, se escogen los valores de **100**, **75** y **50** individuos. La elección de estos 3 valores se debe a que son los que proporcionan los mejores resultados, observando una acusada diferencia con el resto de valores.

- **pC**: A la vista de la Figura 5.3, se escogen los valores de **0.5** y **0.75** para la tasa de cruce. El valor de pC=0.5 ofrece resultados claramente mejores que el resto de valores y pC=0.75 se escoge por ser el que ofrece el segundo mejor resultado, aunque no por demasiada diferencia con el resto.

- **pM**: A la vista de la Figura 5.4, se escoge el valor de **0.01** para la tasa de mutación por ser el que ofrece el mejor resultado. El hecho de escoger solo un valor para pM se debe a que se considera que su efecto no es muy determinante. De esta forma, se considera más enriquecedor escoger 3 valores de número de individuos en lugar de 2 valores de tasa de mutación.

En total, el número de combinaciones es 2x3x2x1=12. Estas **12 combinaciones**, presumiblemente, arrojarán resultados mejores que los obtenidos en el Apartado 5.1. De la misma manera que en la primera tanda de simulaciones del apartado anterior, se han realizado varias repeticiones (en este caso 3) de cada simulación para tratar de disminuir el efecto de la aleatoriedad propia de un algoritmo genético.

En la Tabla 5.5 se muestran las 12 combinaciones realizadas y los resultados obtenidos con cada una de ellas. Las medias aparecen redondeadas a 4 decimales. En las cabeceras de dicha tabla, cabe destacar que NG significa número de generaciones y NI número de individuos.

Combinación	NG	NI	pC	pM	Experimento	*Fitness*
1	100	100	0.5	0.01	1	0.2101
1	100	100	0.5	0.01	2	0.2142
1	100	100	0.5	0.01	3	0.1451
1	**100**	**100**	**0.5**	**0.01**	**Media**	**0.1898**
2	100	100	0.75	0.01	1	0.1783
2	100	100	0.75	0.01	2	0.1130
2	100	100	0.75	0.01	3	0.1618
2	**100**	**100**	**0.75**	**0.01**	**Media**	**0.1510**
3	50	100	0.5	0.01	1	0.1744
3	50	100	0.5	0.01	2	0.1364
3	50	100	0.5	0.01	3	0.1926
3	**50**	**100**	**0.5**	**0.01**	**Media**	**0.1678**
4	100	75	0.5	0.01	1	0.1433
4	100	75	0.5	0.01	2	0.1979
4	100	75	0.5	0.01	3	0.1955
4	**100**	**75**	**0.5**	**0.01**	**Media**	**0.1789**
5	50	100	0.75	0.01	1	0.2015
5	50	100	0.75	0.01	2	0.1867
5	50	100	0.75	0.01	3	0.1848
5	**50**	**100**	**0.75**	**0.01**	**Media**	**0.1910**
6	50	75	0.5	0.01	1	0.1669
6	50	75	0.5	0.01	2	0.1551
6	50	75	0.5	0.01	3	0.1751
6	**50**	**75**	**0.5**	**0.01**	**Media**	**0.1657**
7	100	75	0.75	0.01	1	0.1673
7	100	75	0.75	0.01	2	0.1608
7	100	75	0.75	0.01	3	0.1439
7	**100**	**75**	**0.75**	**0.01**	**Media**	**0.1573**
8	50	75	0.75	0.01	1	0.1589
8	50	75	0.75	0.01	2	0.1559
8	50	75	0.75	0.01	3	0.1549
8	**50**	**75**	**0.75**	**0.01**	**Media**	**0.1566**
9	100	50	0.5	0.01	1	0.1499
9	100	50	0.5	0.01	2	0.1453
9	100	50	0.5	0.01	3	0.1270
9	**100**	**50**	**0.5**	**0.01**	**Media**	**0.1407**
10	50	50	0.75	0.01	1	0.1682
10	50	50	0.75	0.01	2	0.1327
10	50	50	0.75	0.01	3	0.1365
10	**50**	**50**	**0.75**	**0.01**	**Media**	**0.1458**
11	100	50	0.75	0.01	1	0.1265
11	100	50	0.75	0.01	2	0.1525
11	100	50	0.75	0.01	3	0.1685
11	**100**	**50**	**0.75**	**0.01**	**Media**	**0.1492**
12	50	50	0.5	0.01	1	0.1195
12	50	50	0.5	0.01	2	0.2027
12	50	50	0.5	0.01	3	0.1231
12	**50**	**50**	**0.5**	**0.01**	**Media**	**0.1484**

Tabla 5.5: Resultados de las 12 mejores combinaciones

La primera y la quinta combinaciones son las que tienen los *fitness* más altos. Entre ambas, la que parece ser ligeramente mejor es la quinta combinación. Esto puede deberse al mayor porcentaje de cruces que se realizan frente a la primera combinación (un 25 % más). En consecuencia, la combinación que pasa a probarse en el Apartado 5.3 es la quinta, por ser la que ofrece mejores resultados.

A modo ilustrativo, en la Figura 5.5 se puede observar un gráfico de barras con el avance de cada una de las combinaciones del presente apartado.

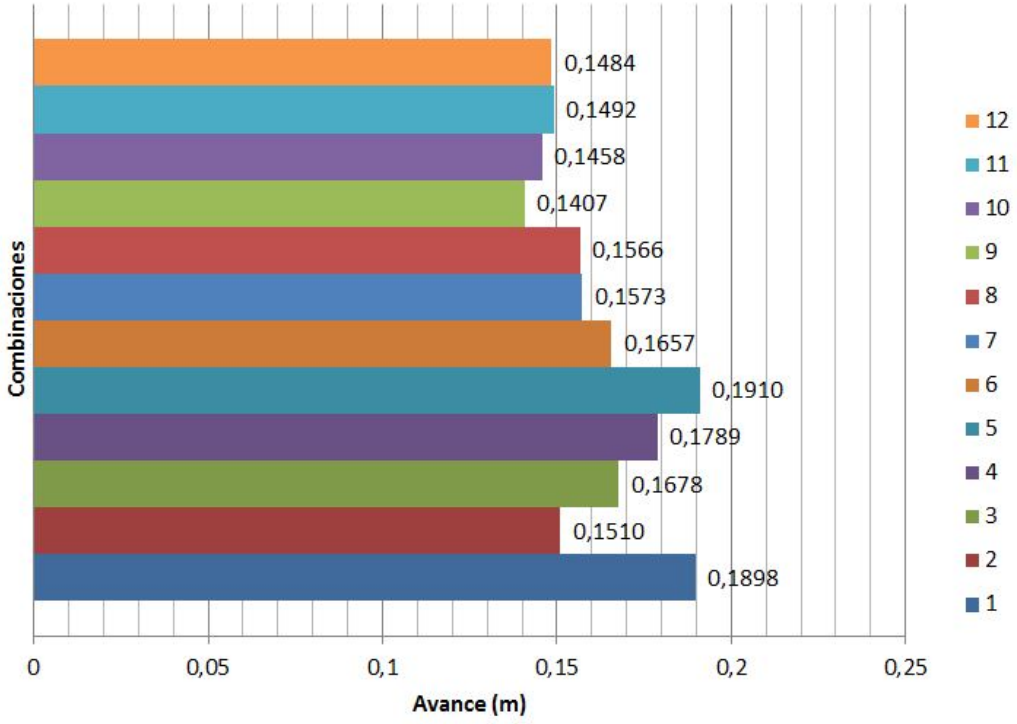


Figura 5.5: *Avance en función de las 12 mejores combinaciones*

Resulta interesante, además, observar la manera en la que el *fitness* evoluciona según se van sucediendo las generaciones dentro del algoritmo genético. Para ello, del primer experimento de cada combinación se ha monitorizado dicha evolución, pudiéndose de esta manera observar tanto la evolución del avance del robot como el momento en el que las soluciones convergen. Nótese que, debido a la naturaleza aleatoria del algoritmo genético, son resultados para el primer experimento de cada combinación, no siendo a la fuerza iguales para el resto de ellos, aunque sí bastante parecidos.

En las siguientes 12 figuras se muestran los resultados. De ellas se puede extraer un análisis interesante, y es que para la quinta combinación (la que, en principio, ofrece mayor *fitness*) se produce la convergencia de la solución en torno a la generación número 24. Mientras que para la

primera combinación (la segunda mejor) se produce la convergencia en torno a la generación 22. Haciendo un símil con la evolución animal, dos generaciones pueden implicar algunas decenas de años que, comparado con los miles de años de evolución biológica en la tierra, puede llegar a ser un tiempo despreciable. Por lo que se desprende que ambas soluciones son muy parecidas, tanto en avance como en tiempo para alcanzar dicho avance.

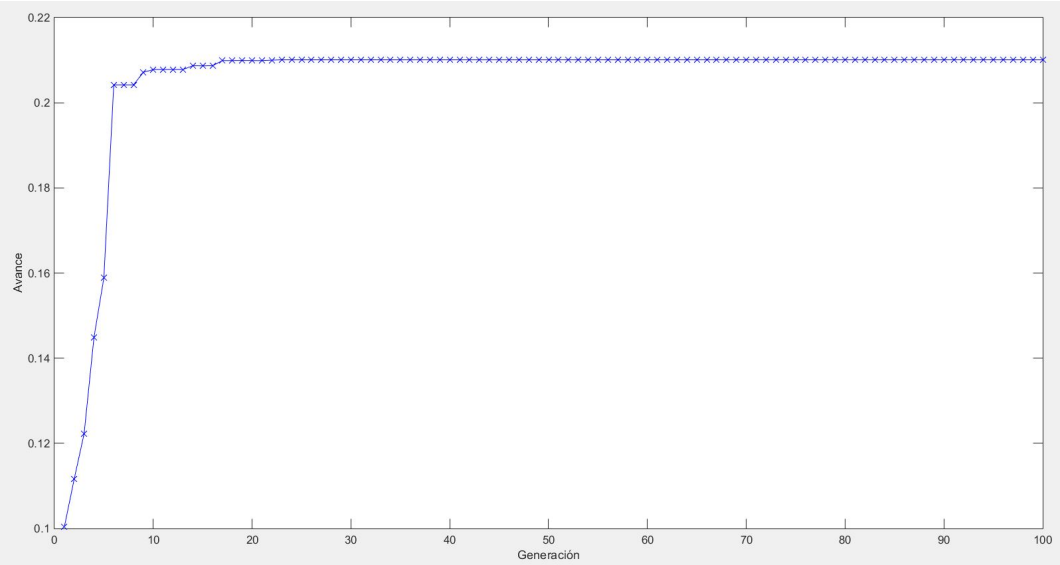


Figura 5.6: *Avance-Generaciones: Combinación 1*

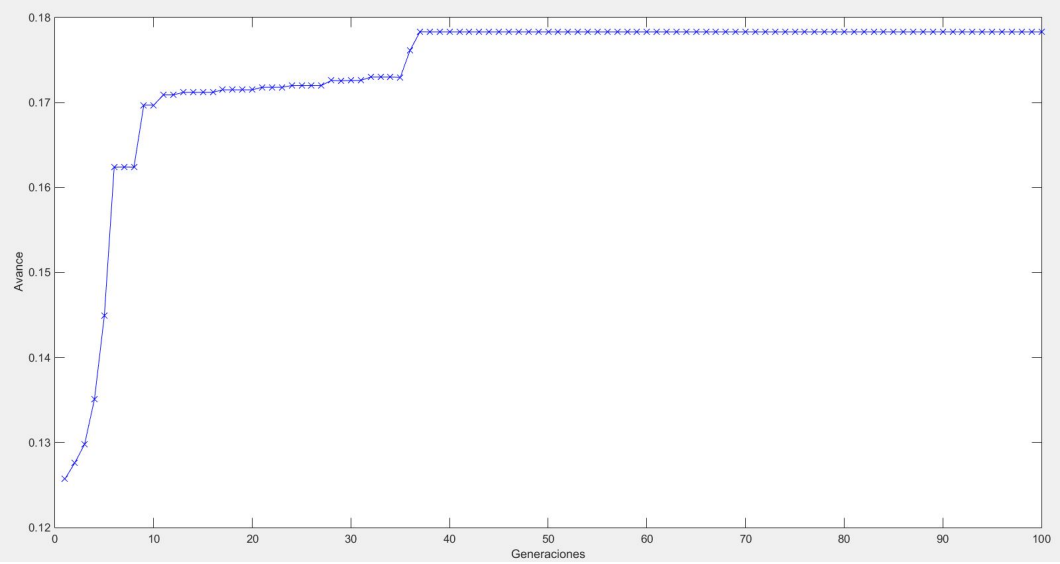


Figura 5.7: *Avance-Generaciones: Combinación 2*

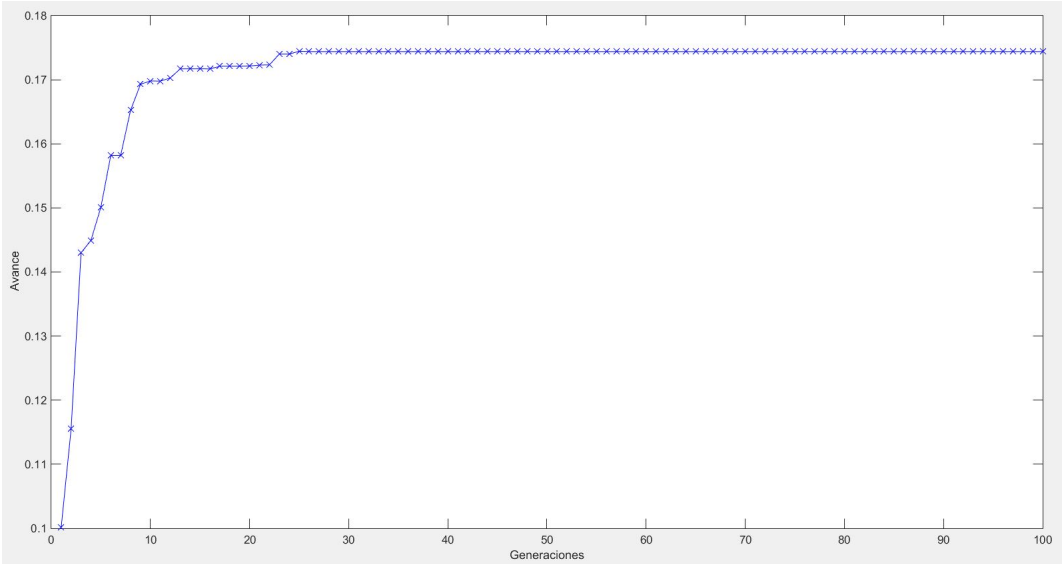


Figura 5.8: *Avance-Generaciones: Combinación 3*

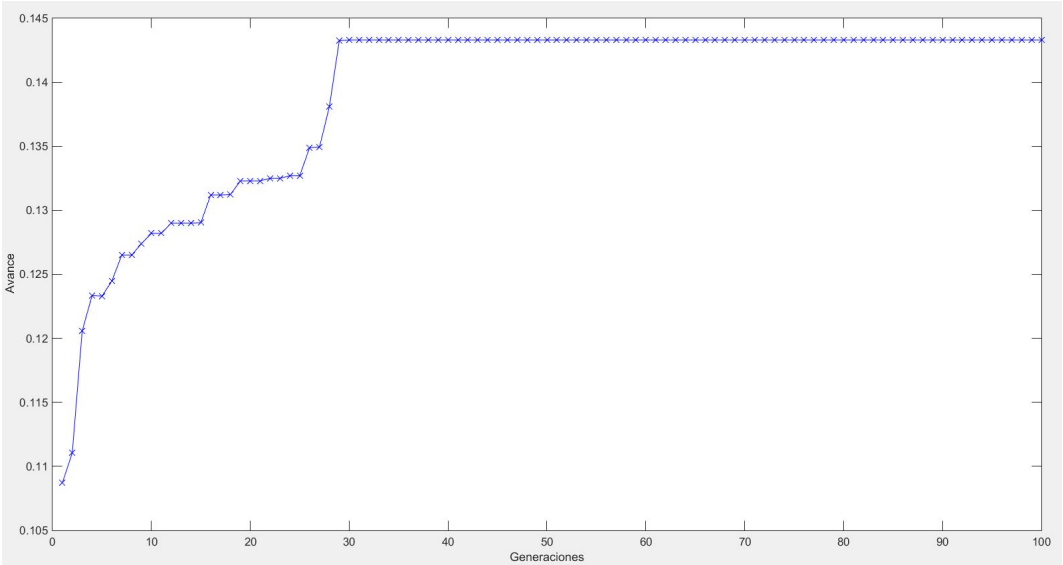


Figura 5.9: *Avance-Generaciones: Combinación 4*

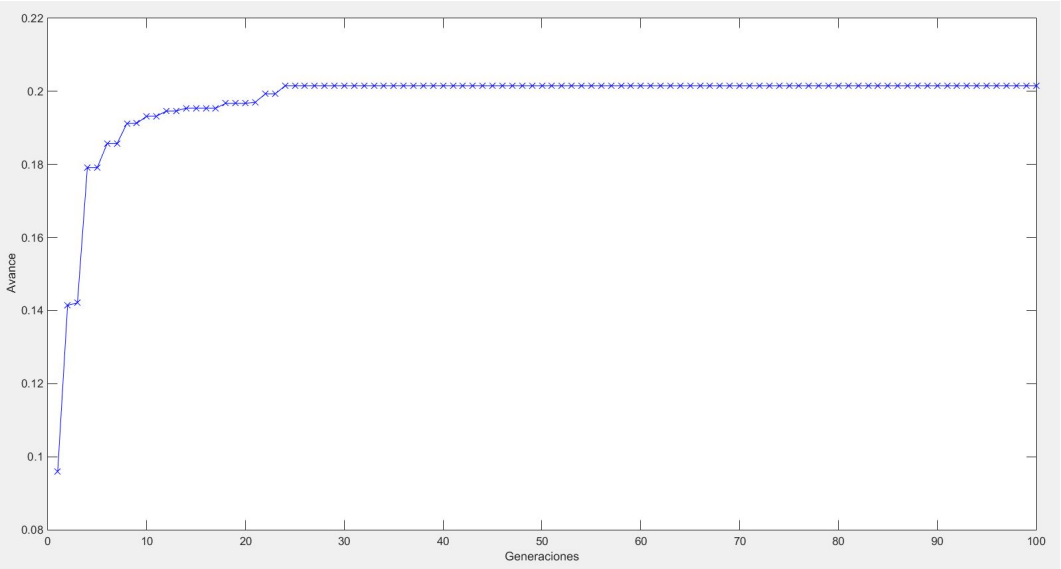


Figura 5.10: *Avance-Generaciones: Combinación 5*

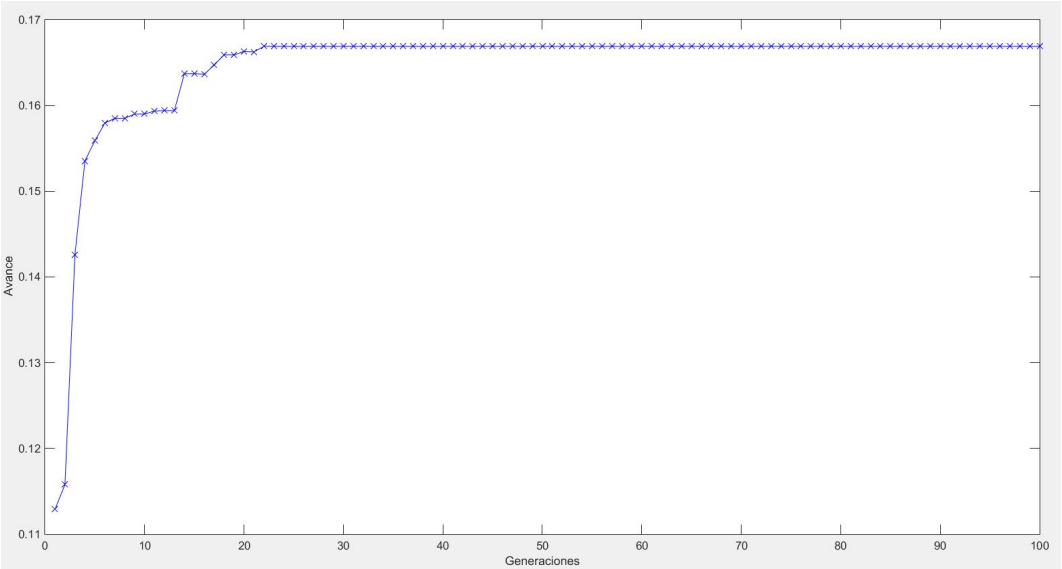


Figura 5.11: *Avance-Generaciones: Combinación 6*

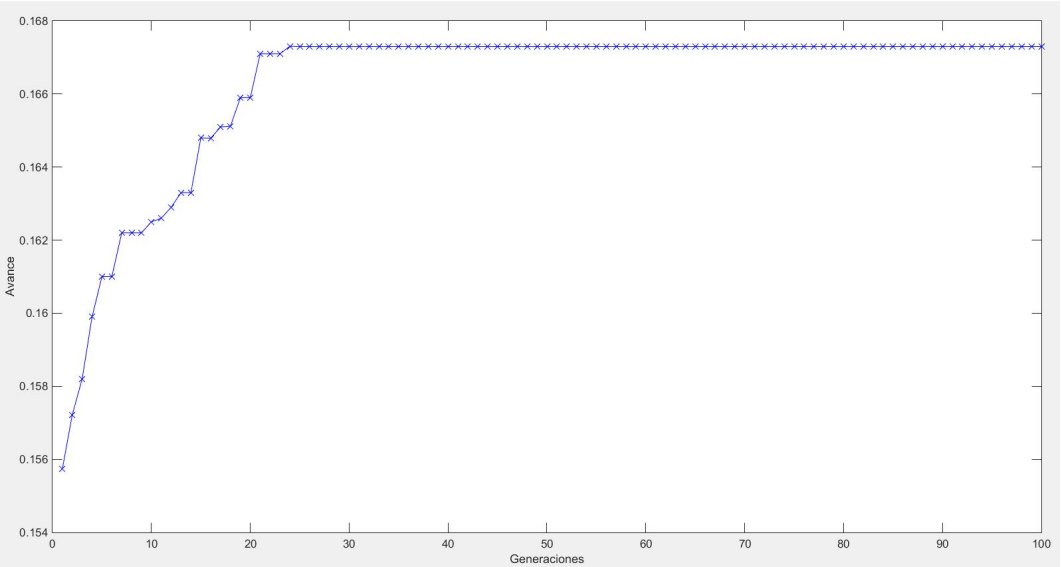


Figura 5.12: *Avance-Generaciones: Combinación 7*

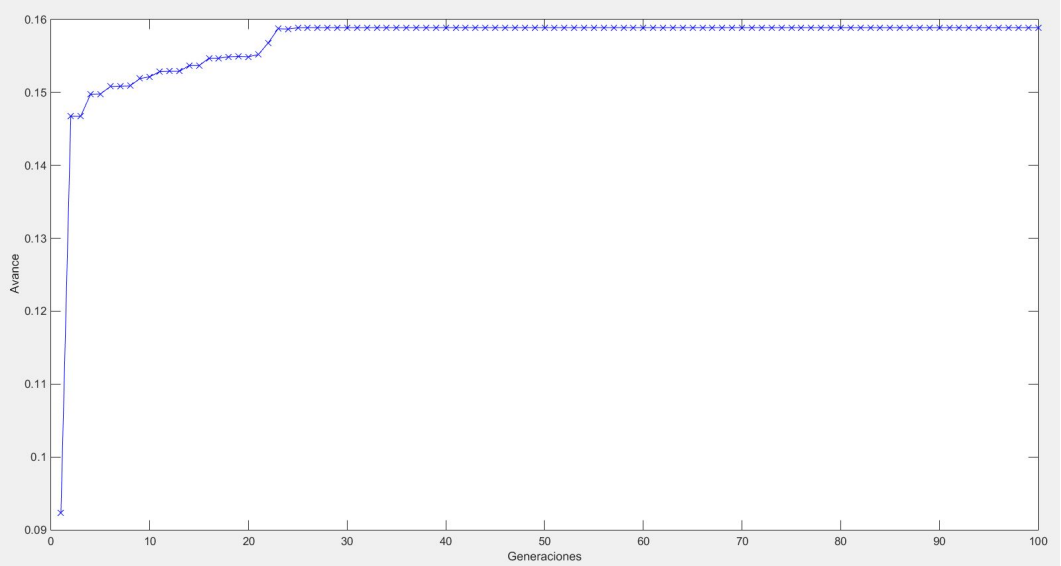


Figura 5.13: *Avance-Generaciones: Combinación 8*

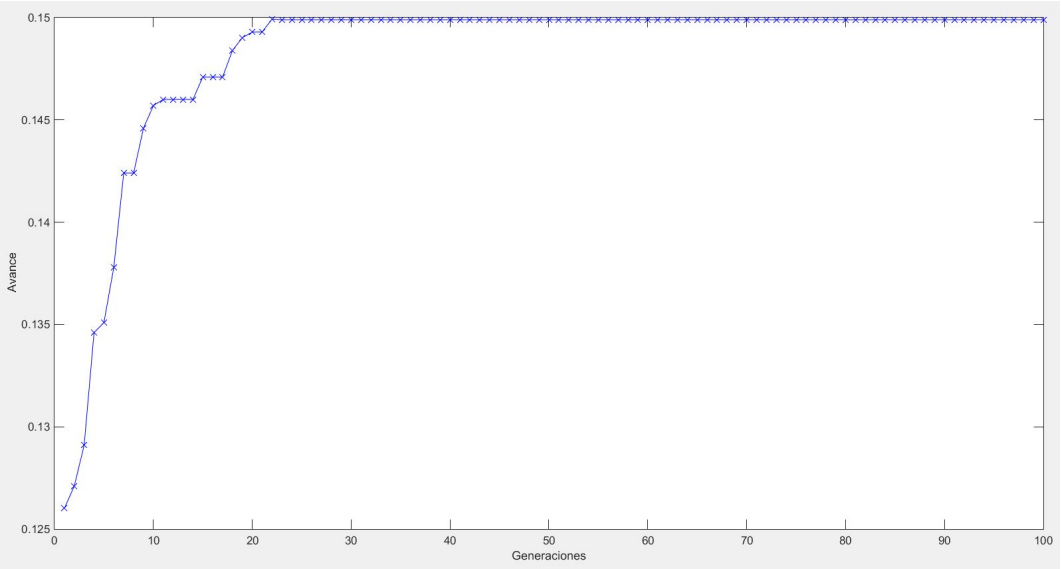


Figura 5.14: *Avance-Generaciones: Combinación 9*

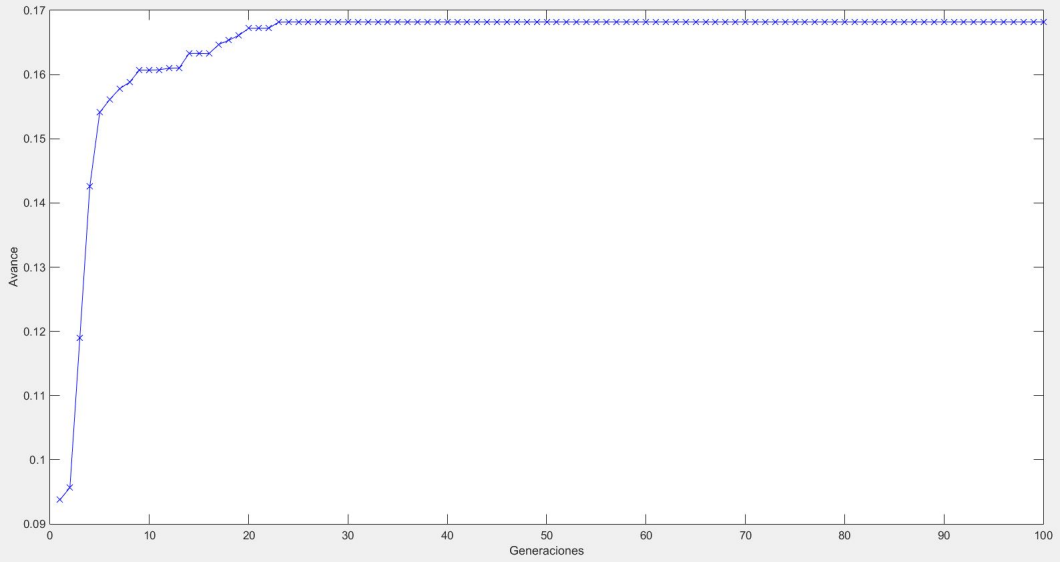


Figura 5.15: *Avance-Generaciones: Combinación 10*

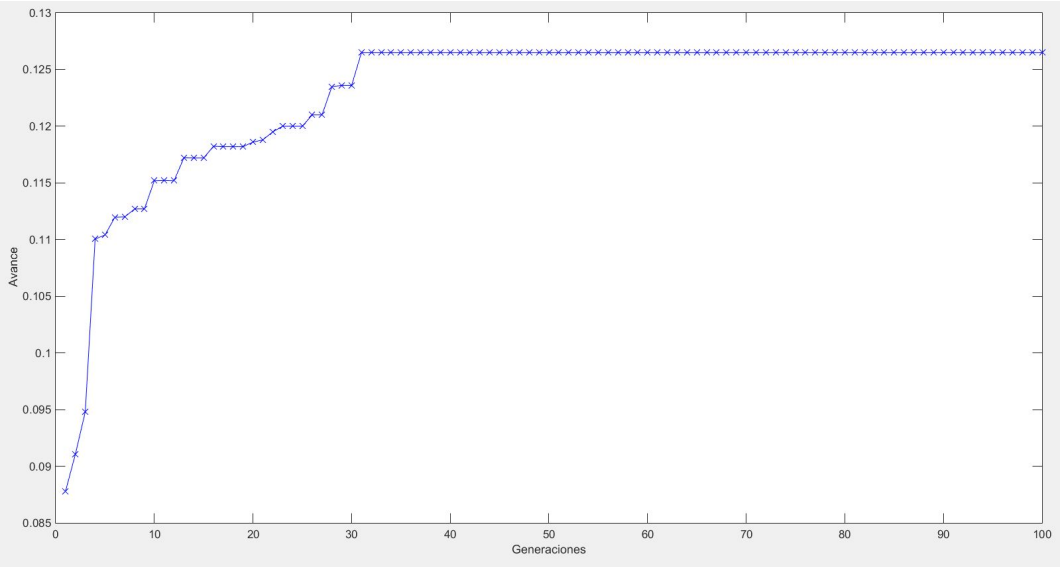


Figura 5.16: *Avance-Generaciones: Combinación 11*

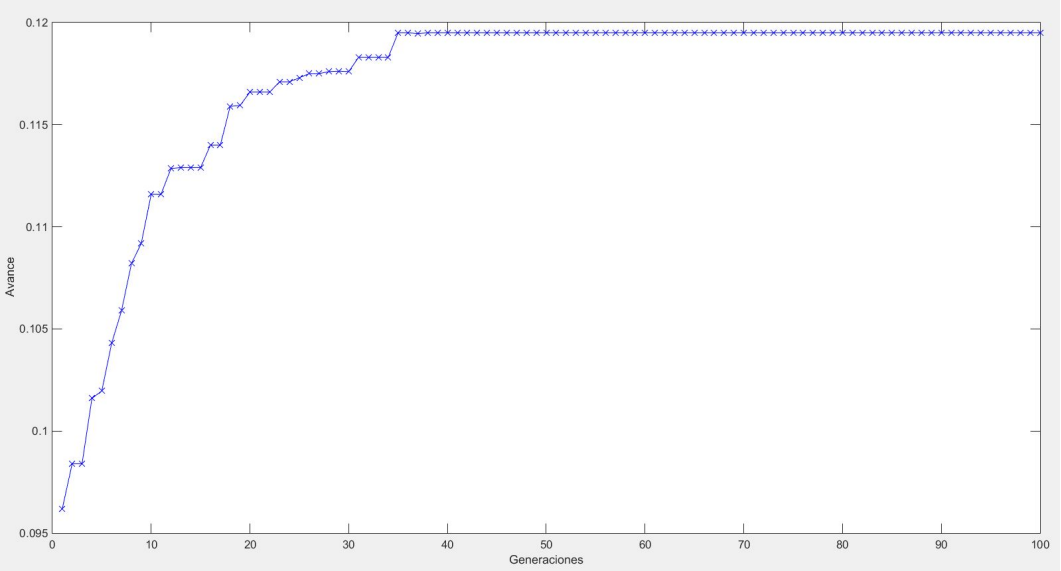


Figura 5.17: *Avance-Generaciones: Combinación 12*

5.3. Variando el cruce y la mutación

En el presente apartado se muestran los resultados de aplicar la mejor combinación de parámetros a otros tipos de cruce y de mutación, distintos a los utilizados en los apartados anteriores.

5.3.1. Otros tipos de cruce

5.3.1.1. Cruce en dos puntos

Con el propósito de realizar pruebas con este tipo de cruce, se ha modificado ligeramente el código de la función **Cruce**. En el Anexo III se puede encontrar el código completo de dicha función. En concreto, se adaptan las líneas de código que contenían el cruce anterior para llevar a cabo un cruce en dos puntos. A continuación se muestran únicamente las líneas de código modificadas.

Líneas de código del cruce en dos puntos:

```
%cruce en dos puntos (tres terceras partes iguales)
aleatorio_tipocruce=ceil(2*rand);
if(aleatorio_tipocruce==1)
        hijo=[padre(1:floor(LongInd/3)) madre((floor(LongInd/3)+1):(floor
            (2*LongInd/3))) padre((floor(2*LongInd/3)+1):end)];
else
        hijo=[madre(1:floor(LongInd/3)) padre((floor(LongInd/3)+1):(floor
            (2*LongInd/3))) madre((floor(2*LongInd/3)+1):end)];
end;
```

En el Apartado 2.5.2.2 se explica el fundamento teórico de este tipo de cruce. Las simulaciones, como ya se ha comentado, se han realizado con la mejor combinación de parámetros de control del Apartado 5.2, que se ha comprobado es la siguiente:

- NumGeneraciones = 50
- NumIndividuos = 100
- pC = 0.75
- pM = 0.01

De la misma manera que en el apartado anterior, se han completado 3 repeticiones y luego se ha calculado su media. Con estos parámetros y este tipo de cruce, los resultados obtenidos se muestran a continuación en la Tabla 5.6. La media ha sido redondeada con 4 decimales.

Experimento	***Fitness***
1	0.1701
2	0.1428
3	0.1601
Media	**0.1577**

Tabla 5.6: Simulaciones con cruce en dos puntos

5.3.1.2. Cruce extendido

Con el propósito de realizar pruebas con este tipo de cruce, se ha modificado ligeramente el código de la función **Cruce**. En el Anexo III se puede encontrar el código completo de dicha función. En concreto, se adaptan las líneas de código que contenían el cruce anterior para llevar a cabo un cruce extendido. A continuación se muestran únicamente las líneas de código modificadas.

Líneas de código del cruce extendido:

```
aleatorio_tipocruce=ceil(2*rand);
alfa=(rand*((5/4)-(-(1/4)))+(-(1/4)));  %constante del cruce extendido para
    ese hijo
if(aleatorio_tipocruce==1)        %cruce extendido
        for k=1:LongInd
                hijo(k)=padre(k)+alfa*(madre(k)-padre(k));
        end;
else
        for k=1:LongInd
                hijo(k)=madre(k)+alfa*(padre(k)-madre(k));
        end;
end;
```

En el Apartado 2.5.2.3 se explica el fundamento teórico de este tipo de cruce. Las simulaciones, como ya se ha comentado, se han realizado con la mejor combinación de parámetros de control del Apartado 5.2, que se ha comprobado es la siguiente:

- NumGeneraciones = 50
- NumIndividuos = 100
- pC = 0.75
- pM = 0.01

De la misma manera que en el apartado anterior, se han completado 3 repeticiones y luego se ha calculado su media. Con estos parámetros y este tipo de cruce, los resultados obtenidos se muestran a continuación en la Tabla 5.7.

Experimento	***Fitness***
1	0.1817
2	0.2082
3	0.1573
Media	**0.1824**

Tabla 5.7: Simulaciones con cruce extendido

5.3.2. Otro tipo de mutación

En el presente apartado se ha probado a simular el algoritmo genético con otro tipo de mutación. En este caso el tipo de mutación utilizada es una en la que se genera una máscara y los genes que correspondan se multiplican por un número aleatorio dentro del intervalo [0,1].

Con objeto de realizar pruebas con este tipo de mutación, se ha modificado ligeramente el código de la función **Cruce** (que es donde, además del cruce, se lleva a cabo la mutación si ésta es necesaria). En el Anexo III se puede encontrar el código completo de dicha función. En concreto, se adaptan las líneas de código que contenían la mutación anterior para llevar a cabo la propuesta en este apartado. A continuación se muestran únicamente las líneas de código modificadas.

Líneas de código de la mutación propuesta:

```
if(pM>aleatorio_mutacion) %mutamos si es necesario
        mascara=round(rand(1,length(hijo))); %genero una mascara aleatoria
            con la que sabre que parametros mutar

         %dos parametros que tengan un 1 en la mascara se multiplican por
             un numero aleatorio en el intervalo [0,1]
        for i=1:(length(hijo))
                if((mascara(i))==1)
                        hijo(i)=rand*hijo(i);
                end;
        end;
end; %fin mutacion
```

Las simulaciones se han realizado con la misma combinación de parámetros de control que en los Apartados 5.3.1.1 y 5.3.1.2. De la misma forma, se han completado 3 repeticiones y luego se ha calculado su media. Con estos parámetros y este tipo de mutación, los resultados obtenidos se muestran a continuación en la Tabla 5.8. La media ha sido redondeada con 4 decimales.

Experimento	***Fitness***
1	0.1937
2	0.1819
3	0.1865
Media	**0.1874**

Tabla 5.8: Simulaciones con otro tipo de mutación

5.4. Análisis de resultados

Una vez han sido realizadas todas las simulaciones, analizando los datos obtenidos se pueden extraer algunas conclusiones:

- Las dos mejores combinaciones de parámetros de control son la **primera** y la **quinta**, que se ha demostrado que ofrecen resultados bastante parecidos tanto en longitud avanzada como en tiempo de algoritmo necesario para llegar a dicha solución. Por consiguiente, son las dos combinaciones que se pasan a las pruebas con el robot real (ver Capítulo 6).

  Combinación **nº1** de parámetros de control:

  - 100 Generaciones
  - 100 Individuos
  - Tasa de Cruce del 50 %
  - Tasa de Mutación del 1 %

  Combinación **nº5** de parámetros de control:

  - 50 Generaciones
  - 100 Individuos
  - Tasa de Cruce del 75 %
  - Tasa de Mutación del 1 %

- El **cruce en un punto** parece ser mejor que los otros dos tipos de cruce con los que se ha trabajado (cruce en dos puntos y cruce extendido). Esta conclusión se puede extraer tras observar que, bajo las mismas condiciones, su media de avance es mayor. En el caso de analizar si es mejor el cruce en dos puntos o el extendido, se puede concluir que los mejores resultados los arroja el cruce extendido.

- De las dos mutaciones con las que se ha trabajado, se han obtenido mejores avances con la **primera mutación** probada (aquella en la que el gen es sustituído por su media entre él mismo y un valor de referencia). El hecho de que con esta mutación se consigan mayores avances implica que es menos "nociva" que la otra opción y, por tanto, al cambiar en menor medida los genes, el *fitness* del individuo se ve menos afectado.

6

Experimentación real del algoritmo

Normalmente el paso de la simulación a la realidad es complejo. Esto se debe a que la realidad es, en la mayoría de casos, más **compleja e impredecible** de lo que nos gustaría. En el presente proyecto dicho salto de simulación a realidad pasa además por el aprendizaje del control del robot mediante **Arduino**, que no se va a tratar en este texto.

Durante los experimentos reales se han grabado en vídeo los resultados. Debido a la evidente imposibilidad de mostrar dichos vídeos en el presente documento, lo que se procede es a mostrar secuencias de imágenes extraídas de dichas grabaciones.

En este capítulo se abordan los experimentos que se han realizado con el robot real. Se han trasladado los resultados obtenidos en las simulaciones del Capítulo 5 y se han probado en la realidad. Se han probado, por tanto, **4 casos**. En el Apartado 6.1.1 se ilustran los resultados de mover el robot con el **peor individuo** posible y en el Apartado 6.1.2 se muestra qué ocurre cuando se mueve el robot con un **individuo intermedio** (aquel que no es el peor pero tampoco el mejor). Por último, en el Apartado 6.1.3, se explican los resultados de aplicar al robot real los dos **mejores individuos** obtenidos en el capítulo anterior, que como ya se ha explicado en el Apartado 5.4 son la **primera** y la **quinta** combinación. Para finalizar, en el Apartado 6.2 se realiza un breve análisis de los resultados obtenidos en dichos experimentos reales.

En la Figura 6.1 se muestra el robot cuadrúpedo real con el que se ha trabajado.

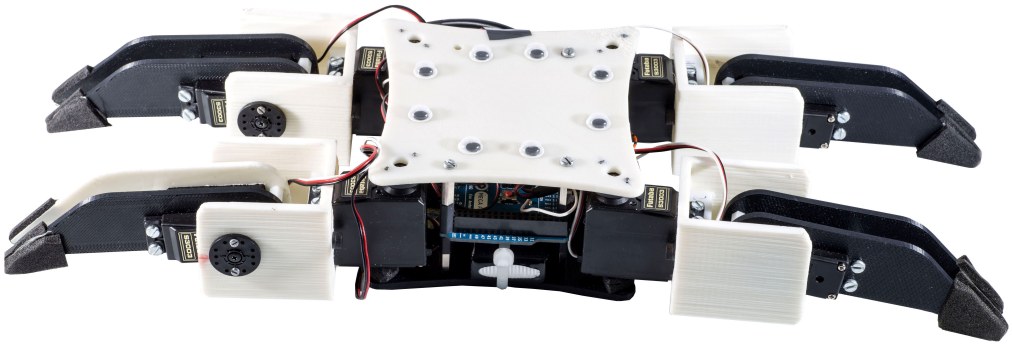


Figura 6.1: *Robot cuadrúpedo real*

6.1. Experimentos realizados

6.1.1. Peor solución

La peor solución es aquel individuo que tiene el *fitness* más bajo, es decir, aquel individuo cuya combinación de parámetros ofrece un avance precario y reducido.

En la secuenia de imágenes de la Figura 6.2 se puede observar que el robot no consigue pasar de la primera línea concéntrica que lo rodea y se limita a dar vueltas en círculos sobre sí mismo. Para observar la secuencia de imágenes en el orden correcto hay que verlas de izquierda a derecha y de arriba abajo.

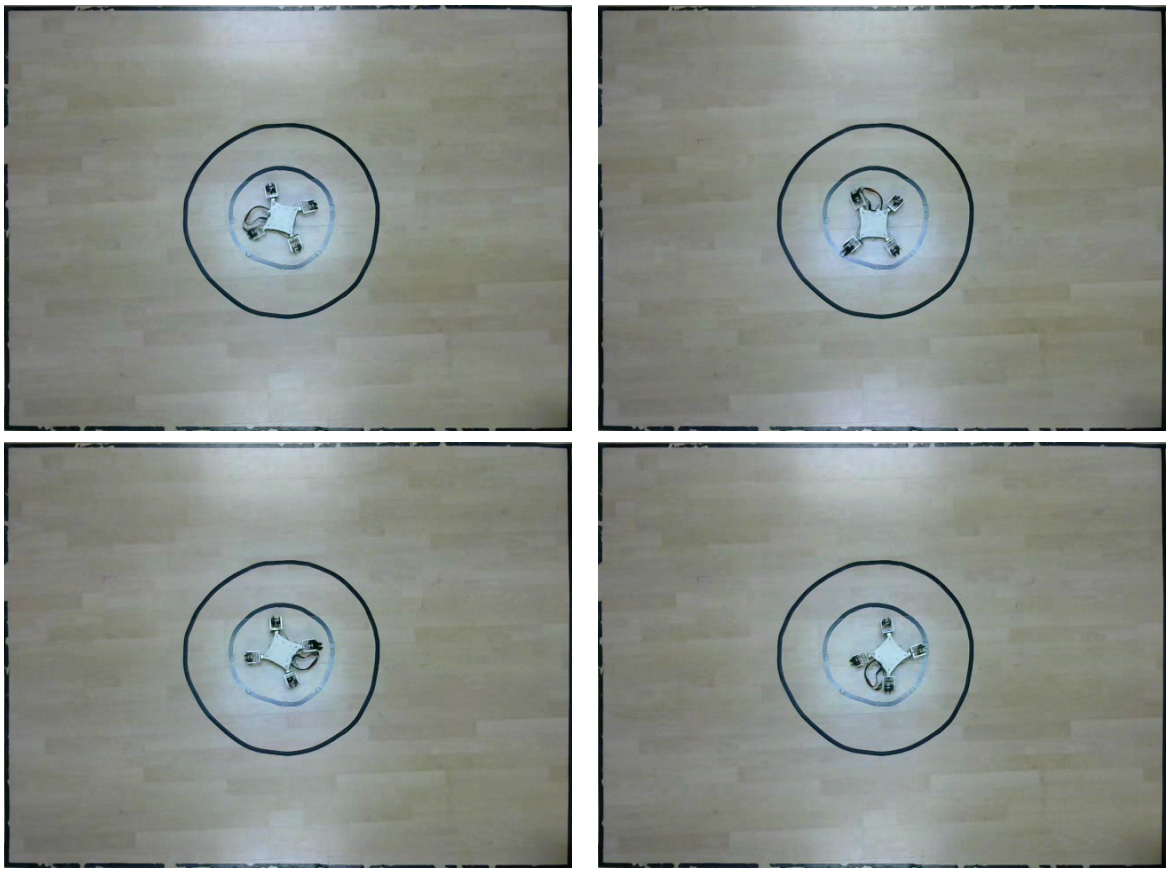


Figura 6.2: *Secuencia de avance del peor individuo*

Con el afán de comprender mejor el motivo por el cual el peor individuo se comporta de esta manera se puede tratar de observar más de cerca los distintos movimientos que realiza el robot. De esta manera, es posible extraer la secuencia y comprender cuál o cuáles son los movimientos que hacen al robot avanzar poco y en círculos. En las imágenes de la Figura 6.3 se ilustra el movimiento perjudicial para el avance.

En la imagen izquierda, se observa como el robot abre dos patas de la misma diagonal al mismo tiempo y acto seguido, en la imagen derecha, las cierra a la vez. Esta combinación no consigue un avance significativo debido a que cada pata tratará de avanzar en direcciones opuestas y sus movimientos se compensarán. Este movimiento junto con el de la otra diagonal, que se puede observar en la secuencia de imágenes de la Figura 6.4 (de izquierda a derecha y de arriba abajo), produce que el robot gire sobre sí mismo.

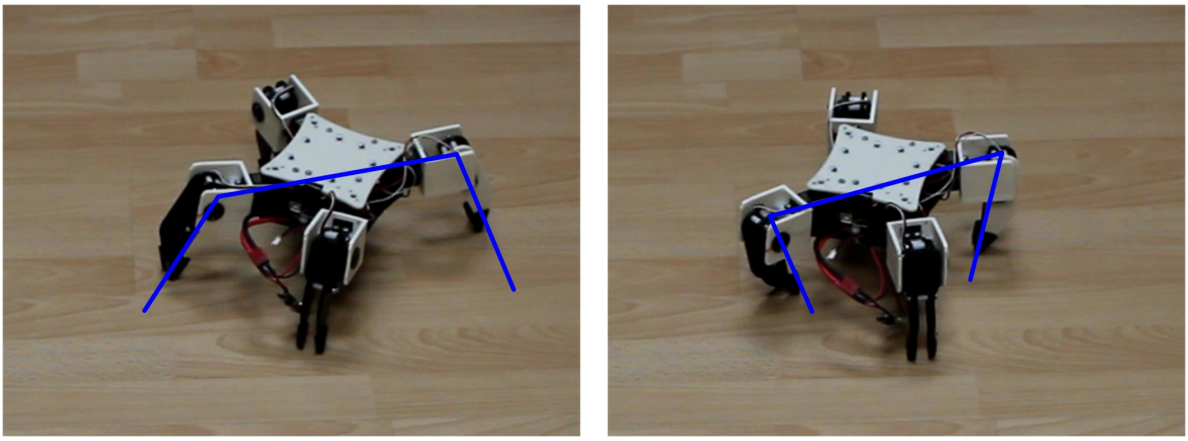


Figura 6.3: *Movimiento perjudicial para el avance*

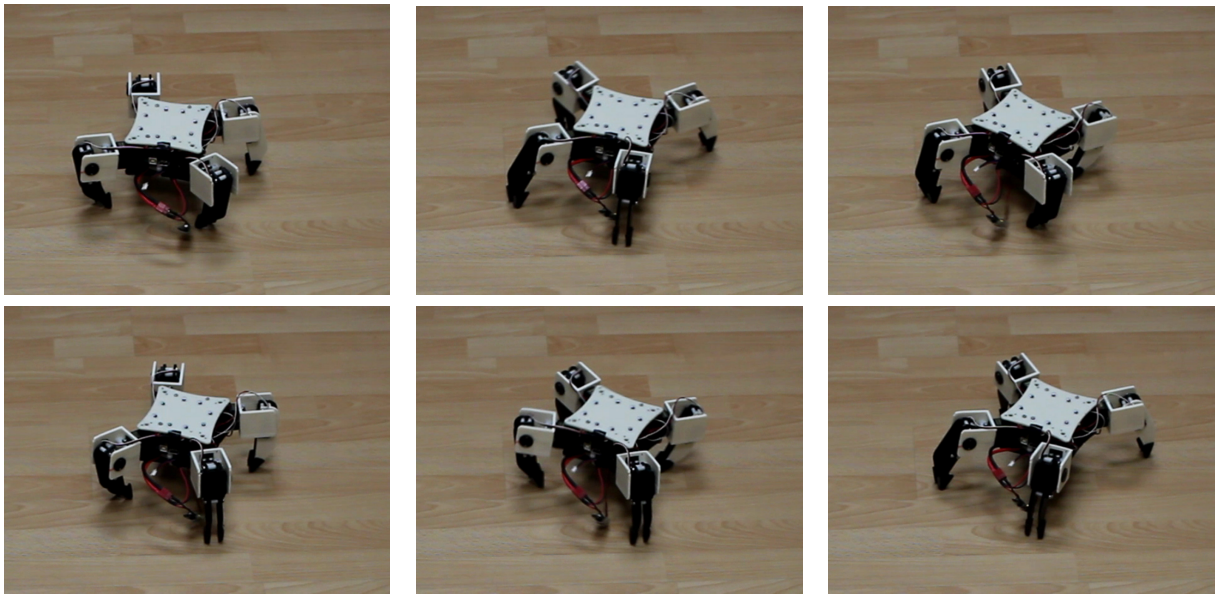


Figura 6.4: *Secuencia de avance perjudicial*

6.1.2. Solución intermedia

Por solución intermedia se entienden todos aquellos individuos cuyas soluciones están entre las peores y las mejores, es decir, aquellos individuos que consiguen un avance intermedio. En el presente apartado se han realizado pruebas con uno de estos individuos, obtenido de las simulaciones.

En la secuencia de imágenes de la Figura 6.5 se observa que el robot avanza mejor que en el caso del Apartado 6.1.1, pues consigue sobrepasar la primera línea concéntrica y llegar hasta la segunda. Dichas líneas se encuentran separadas $10cm$. Para observar la secuencia de imágenes en el orden correcto hay que verlas de izquierda a derecha y de arriba abajo.

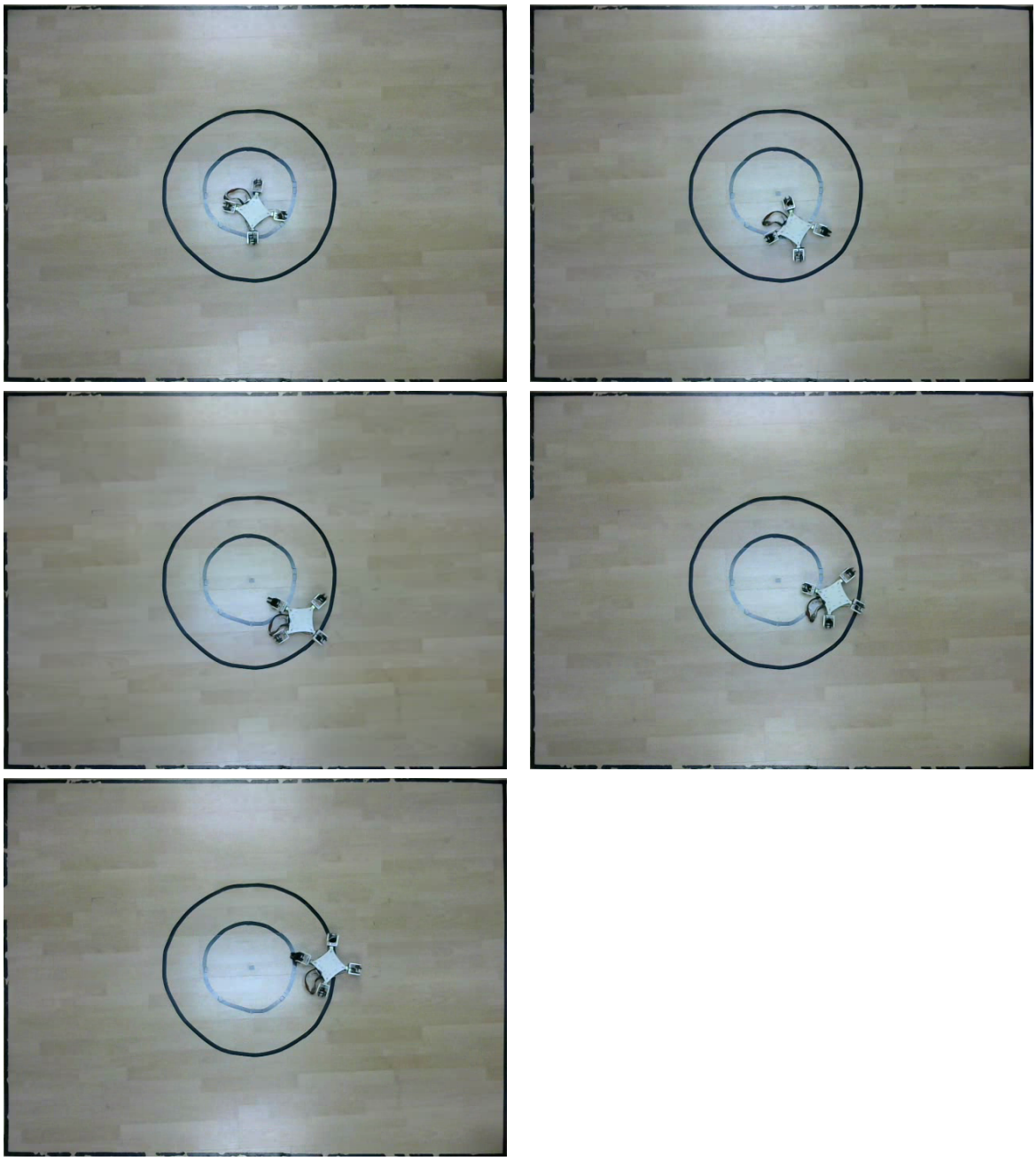


Figura 6.5: *Secuencia de avance de un individuo intermedio*

En caso de querer comprender por qué el robot avanza una mayor distancia con este individuo, se puede analizar la secuencia de movimiento más de cerca. Si se observa con atención la secuencia de imágenes de la Figura 6.6 (de izquierda a derecha y de arriba abajo) se puede extraer que las patas 1 y 4 apenas son útiles, pues apenas se mueven. Por otra parte, las patas 2 y 3 realizan movimientos que consiguen hacer avanzar mejor el robot, pues se van estirando y contrayendo alternativamente.

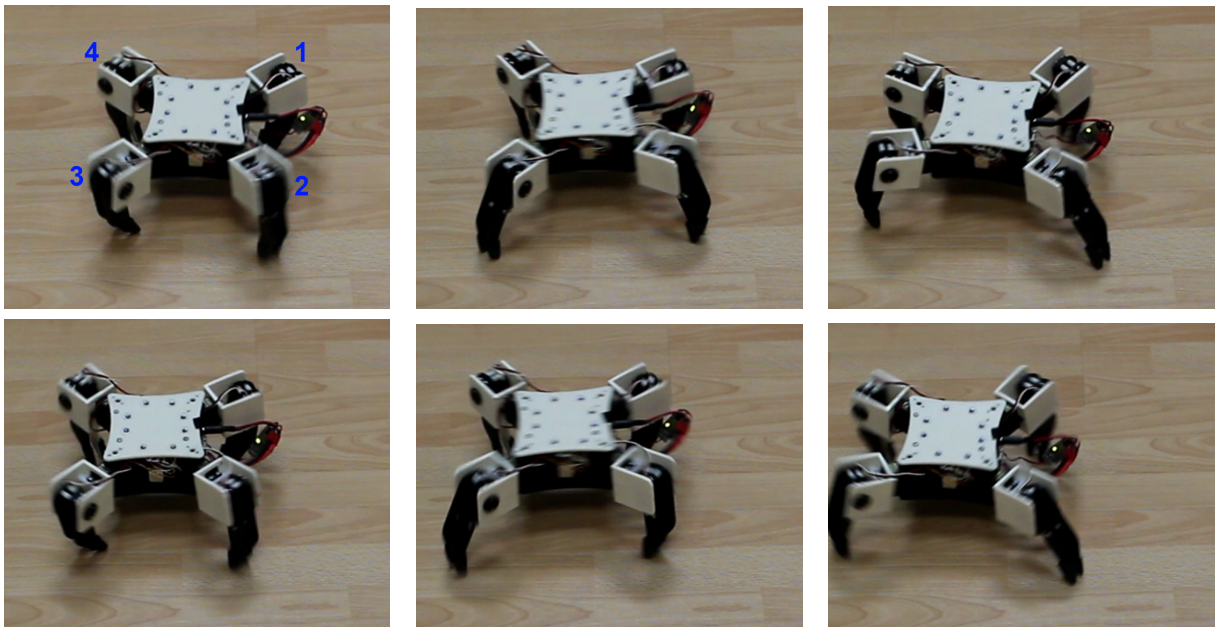


Figura 6.6: *Secuencia de avance intermedio*

6.1.3. Mejores soluciones

Cuando se habla de mejores soluciones, se quiere hacer referencia a las **mejores soluciones obtenidas en la simulación**. Es evidente que, cuando se trata de algoritmos evolutivos, **no** se puede hablar de mejor solución como aquella que es inmejorable. En términos naturales sería como decir que una especie animal ya no puede evolucionar más. Por tanto, con mejor solución se hace referencia a **mejor solución de las que se han barajado y simulado**; habiendo siempre, pues, lugar para la mejora y la evolución.

Las soluciones con mejores resultados en las simulaciones son la primera y la quinta combinación; queriéndose comprobar si el avance es mayor que en los casos de los dos apartados anteriores, donde se han probado la peor solución y una intermedia. En la secuencia de imágenes de la Figura 6.7 se muestra el avance de un individuo con la quinta combinación y en la Figura 6.8 el avance de un individuo con la primera combinación. Para observar las secuencias de imágenes en el orden correcto hay que verlas de izquierda a derecha y de arriba abajo.

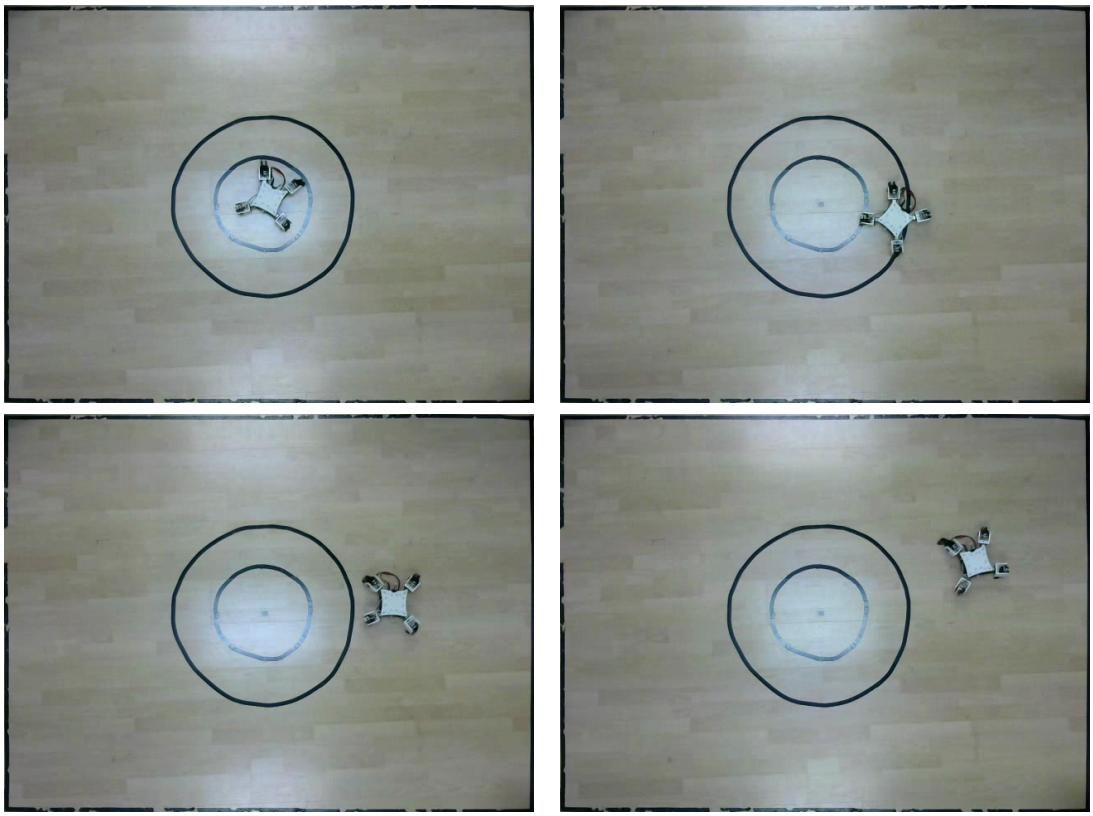


Figura 6.7: *Secuencia del mejor individuo (quinta combinación)*

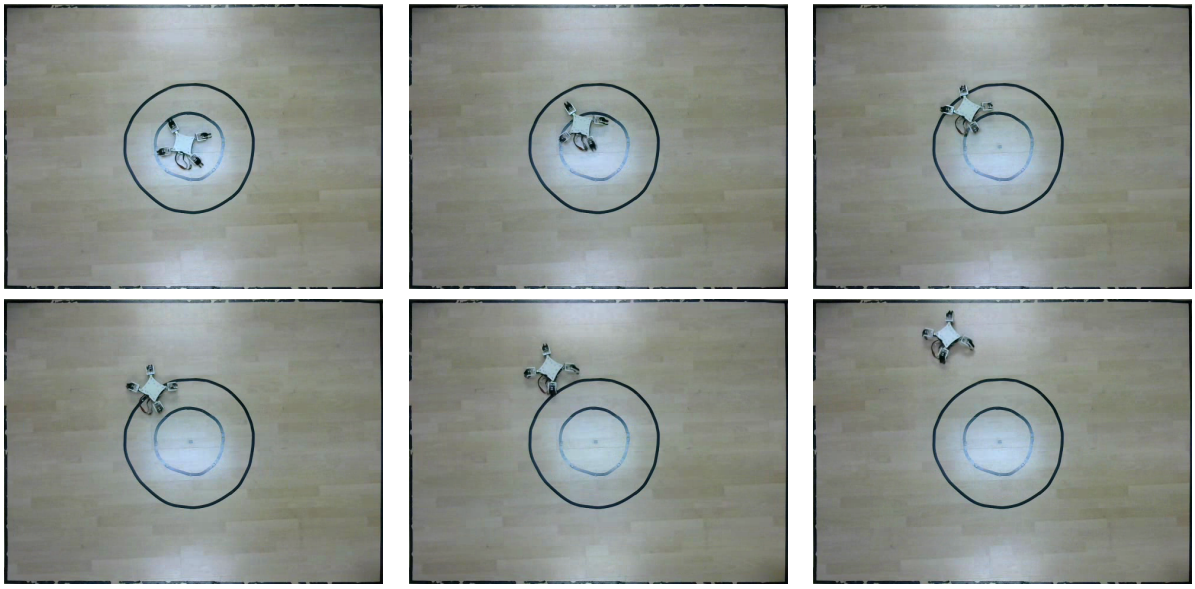


Figura 6.8: *Secuencia del mejor individuo (primera combinación)*

De la misma forma que en los apartados anteriores se han tratado de analizar los movimientos que realiza el robot más de cerca, se realiza la misma operación aquí. Lo que se persigue es descubrir aquellos movimientos beneficiosos que hacen que el avance sea mayor en el robot. Para ello se hace uso de la numeración de las patas que se muestra en la Figura 6.9.

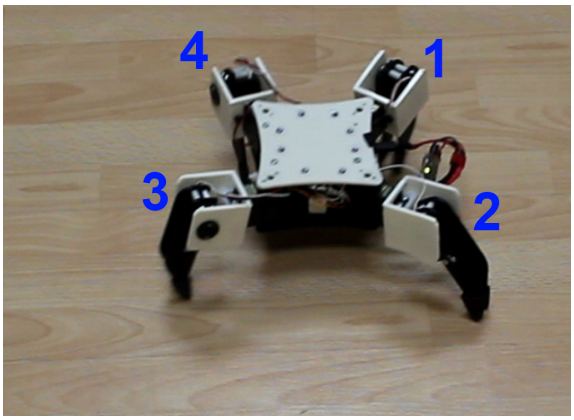


Figura 6.9: *Numeración de las patas*

En la Figura 6.10 se muestran los movimientos que hacen mejores a estas dos soluciones. Al contrario que ocurría en el peor caso, las patas se abren y cierran disparmente produciendo un desplazamiento beneficioso en una dirección. En las imágenes se puede ver como en las diagonales formadas por las patas 1 y 3 y por las patas 2 y 4, se produce una apertura de una de las patas de la diagonal al mismo tiempo que la otra se contrae. En el siguiente movimiento la que estaba contraída se abre y vicerversa.

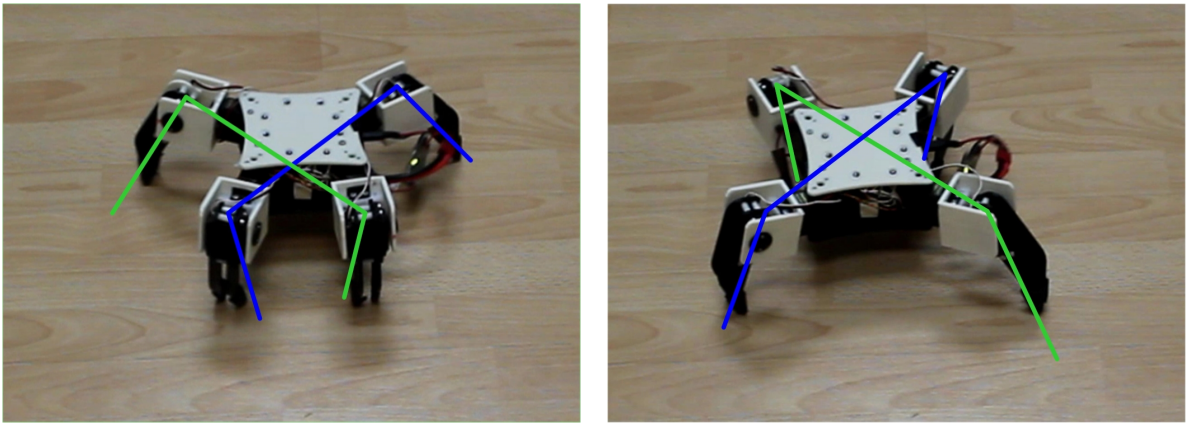


Figura 6.10: *Movimiento beneficioso para el avance*

Por último, se muestra a continuación la secuencia al completo. Para observar las secuencias de imágenes en el orden correcto hay que verlas de izquierda a derecha y de arriba abajo.

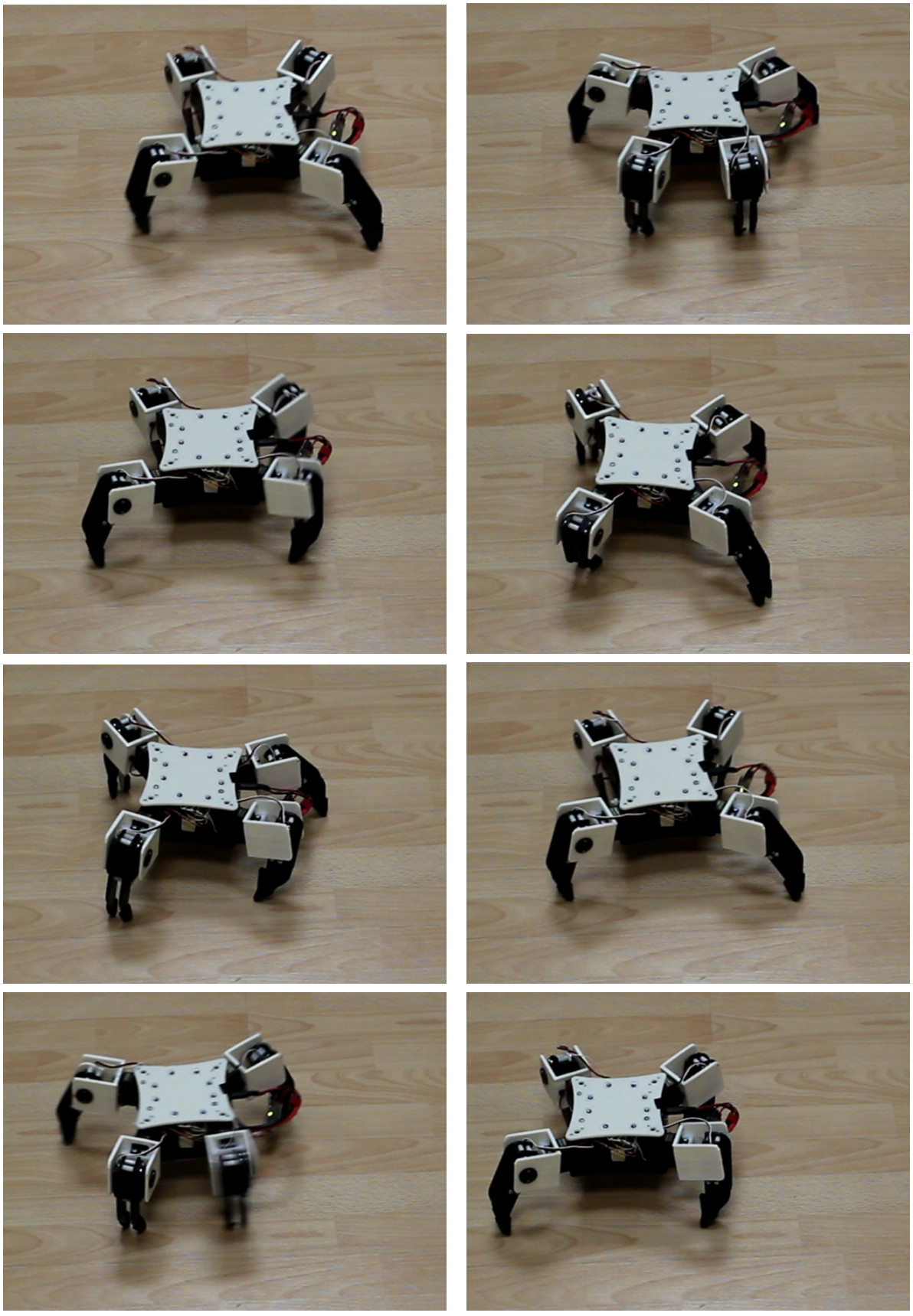


Figura 6.11: *Mejor secuencia de avance*

6.2. Análisis de resultados

Los resultados de los experimentos con el robot real son **satisfactorios**. Se ha podido comprobar que efectivamente los **mejores individuos** obtenidos en las simulaciones del Capítulo 5 producen un **avance mayor al resto**.

El **peor individuo** estudiado se ha comprobado que **no realiza un avance significativo**, si no que lo que hace es dar vueltas sobre sí mismo. Por consiguiente, puede observarse una **evolución del avance** en la realidad a medida que las soluciones van mejorando en la simulación. También ha sido posible analizar de manera exitosa los distintos **movimientos buenos y malos** que hacen avanzar al robot de una determinada manera u otra.

Por otra parte, cabe destacar que sería interesante realizar una monitorización de la secuencia de avance del robot a medida que **van pasando las generaciones** para una solución dada. También es importante comentar que los experimentos realizados con la cinta aislante en el suelo han dado algunos problemas, pues en ocasiones las patas del robot **se quedan atrancadas con la cinta** y no puede avanzar como debería. Esto quizá podría solucionarse **pintando los círculos** en lugar de utilizar cinta aislante.

Otro punto destacable, y que por falta de tiempo no ha sido posible llevar a cabo, es tratar de realizar estos mismos experimentos **en otras superficie distintas**. De esta manera podría analizarse la **influencia del medio** en las soluciones. Por último, podría tratarse de **inhabilitar alguno de los motores** en el robot y correr el algoritmo para analizar la secuencia a la que se llegaría entonces.

7

Conclusiones y líneas futuras

7.1. Conclusiones

El presente trabajo, titulado “Sistema de aprendizaje evolutivo para un robot caminante”, se encuentra dentro de la línea de trabajo de algoritmos evolutivos, inmerso en el Departamento de Automática, Ingeniería Electrónica e Informática Industrial de la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid. En él se ha investigado y trabajado acerca de los algoritmos evolutivos, consiguiendo que el robot aprenda a caminar utilizando dichos sistemas de aprendizaje.

Para ello ha sido necesaria, al comienzo, una profunda investigación sobre el estado del arte. Se ha llevado a cabo con éxito una recopilación de información útil, ordenada y sencilla de seguir. Además, se considera que lo aprendido tras este trabajo es muy valioso e importante, pues se puede aplicar a infinidad de situciones y problemas.

El modelo de simulación desarrollado se ha comprobado que es fiable y, además, guarda un estrecho parecido con el robot real, pudiendo así simular la realidad con bastante similitud. En la parte evolutiva se ha conseguido codificar con éxito un algoritmo evolutivo con el que se puede investigar dándole diferentes parámetros para obtener distintos resultados. Dicho algoritmo consigue que el robot sea capaz de aprender a caminar. Cabe destacar la importancia que tiene la correcta elección de la codificación de los individuos, así como la función de coste o función de *fitness*.

La experimentación ha sido existosa, pues se han podido trasladar los resultados de las simulaciones a la realidad. Se ha comprobado que se produce un avance creciente del robot, a medida que las soluciones van siendo mejores. También se han podido observar distintos tipos de movimientos que llevan al robot a desplazarse en mayor o menor medida.

Durante el citado proyecto se demuestra el poder que ofrece la conjunción del aprendizaje evolutivo con los programas informáticos de simulación robótica. Adicionalmente, ha quedado patente el amplio rango de aplicación de estos sistemas, pudiendo de esta manera simular cualquier problema de aprendizaje robótico y optimizar tiempo y recursos. Del desarrollo de este trabajo se desprende la importacia del aprendizaje robótico, ya que es posible conseguir que la róbótica se adapte en cualquier situación, haciéndonos de esta forma la vida más sencilla.

La evolución de la naturaleza, y más concretamente la evolución del ser humano, hace totalmente necesario llevar a cabo proyectos en los cuales la evolución esté presente. Se hace notorio que en el afán del ser humano de tomar ejemplo de la sabia naturaleza, el aprendizaje en el mundo de los robots toma un papel de suma importancia al brindar la oportunidad de construir un mundo adaptado a nuestras necesidades.

7.2. Líneas futuras de actuación

A lo largo del desarrollo del presente Trabajo Fin de Grado se han visualizado diferentes líneas de actuación, con las que mejorar y ampliar lo conseguido. Estas mejoras, debido a la visión generalista de los Trabajos Fin de Grado y al tiempo disponible, no han podido ser llevadas a cabo. Por lo tanto, partiendo de este TFG como punto de partida, se pueden enfocar otros trabajos bajo las siguientes líneas actuación:

- Observar la **evolución de la secuencia de avance con el paso de las generaciones**. De esta manera podría investigarse la manera en la que va mejorando la secuencia de paso a medida que se van sucediendo las generaciones.
- **Mejorar el modelo de simulación**. Sería interesante realizar un análisis sobre las físicas del modelo y el rozamiento existente entre las patas y el suelo.
- Relacionado con el punto anterior, podría pensarse en **cambiar de programa de simulación robótica** para cambiar de esta forma el proceso de simulación, buscando una **mayor eficiencia de tiempo en cada simulación**.
- **Mayor amplitud de investigación acerca del Algoritmo Genético**. Sería interesante probar más tipos de cruce, de mutación y más combinaciones de parámetros de control. De esta manera, se amplía el abanico de búsqueda de soluciones óptimas.
- Probar **otros métodos de aprendizaje** distintos a los algoritmos evolutivos, como por ejemplo redes neuronales.
- Probar **otra función de *fitness***. El avance, sin más, puede ser en ocasiones engañoso debido a que el robot puede dar vueltas en círculos porque no se mide el avance en una dirección concreta. Podría tratarse de mejorar la medida del avance o pensar en otras funciones de coste como por ejemplo que el robot no se tambalee demasiado durante la marcha.
- Probar **otra codificación de individuos**. Mediante el cambio de la codificación de individuos quizá podría llegarse a obtener mejores resultados. Podría utilizarse señales cuadradas o triangulares, en lugar de una senoide.
- Realización de **pruebas con el robot real más extensas**. Podría probarse la manera de avanzar del robot en diferenetes superficies como tierra o hierba, y no solo en superficies lisas. Así mismo, sería interesante ver lo que ocurre si se inhabilita uno o varios motores del robot, simulando de esta manera que se ha producido un daño. Por último, también sería interesante tratar de superar obstáculos.
- **Mejorar el diseño o los materiales del robot**. Podría pensarse en utilizar más patas, más grados de libertad y/o mejorar los materiales. Por otra parte sería beneficioso para la estética del robot, tratar de embeber mejor los cables para que no se encuentren tan sueltos.

Bibliografía

[1] CULLY, A.,MOURET, J. -B. *(2015)*, *Evolving a Behavioral Repertoire for a Walking Robot*, Evolutionary Computation Journal, Vol 1, 1-33.

[2] KOOS, S.,CULLY, A.,MOURET, J. -B. *(2013)*, *Fast Damage Recovery in Robotics with the T-Resilience Algorithm*, International Journal of Robotics Research, Vol 32, No 14, 1700-1723.

[3] SEVERAL DAMAGE RECOVERY SCENARIOS WITH THE T-RESILIENCE ALGORITHM, *[Web en línea]*, <*https://youtu.be/dncuBUnfkA4*>, [Consulta: 15-02-2015]

[4] BONGARD, J.,ZYKOV, V.,LIPSON, H. *(2006)*, *Resilient Machines Through Continuous Self-Modeling*, Science, Vol 314, No 5802, 1118-1121.

[5] RESILIENT ROBOT, *[Web en línea]*, <*https://youtu.be/3HFAB7frZWM*>, [Consulta: 15-02-2015]

[6] FOGEL, D. B. *(2000)*, *What is evolutionary computation?*, Spectrum, IEEE, 37(2), 26, 28-32.

[7] FOGEL, D. B. *(2006)*, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, NJ. Third Edition.

[8] DARWIN, C. *(2007)*, *Descent of Man*, NuVision Publications, ISBN: 9781595478863.

[9] MCCULLOCH, W. S., PITTS, W. H. *(1943)*, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bulletin of Mathematical Biophysics, Vol. 5, 115-133.

[10] YANG, X. -S. *(2010)*, *Engineering Optimization: an introduction with metaheuristic applications*. University of Cambridge. Published by John Wiley & Sons, Inc., Hoboken, New Jersey. ISBN: 978-0-470-58246-6.

[11] BELLERÍN, C.,GARCÍA, E. *(2013)*, *Diseño e implementación de un cuadrúpedo que "aprende" a caminar mediante Algoritmos Genéticos*, Memoria Laboratorio Robótica, Escuela Técnica Superior de Ingenieros Industriales de Madrid (UPM).

[12] DARWIN, C. *(1859)*, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*.

[13] WHITLEY, D. *(1994)*, *A genetic algorithm tutorial*. Computer Science Department, Colorado State University.

[14] BÄCK, T. *(1996)*, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, USA.

[15] TOMASSINI, M. *(1995)*, *A survey of genetic algorithms*. Annual Reviews of Computational Physics, Vol. 3, 87-118.

[16] HOLLAND, J. H. *(1975)*, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor. Republished by the MIT Press (1992).

[17] SMITH, D. E.,GENESERETH, M. R. *(1985)*, *Ordering Conjunctive Queries*. Artifcial Intelligence 26:171-215.

[18] FUJIKI, C. *(1986)*, *An Evaluation of Holland's Genetic Algorithm Applied to a Program Generator*. M.S. thesis, Department of Computer Science, Moscow, ID: University of Idaho.

[19] HICKLIN, J. F. *(1986)*, *Application of the Genetic Algorithm to Automatic Program Generation*. M.S. thesis, Department of Computer Science. Moscow, ID: University of Idaho.

[20] FOGEL, L. J.,OWENS, A. J.,WALSH, M. J. *(1966)*, *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, New York.

[21] RECHENBERG, T. *(1971)*, *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Tesis doctoral. Reimprimido por Fromman-Holzboog (1973).

[22] SCHWEFEL, H. -P. *(1974)*, *Numerische Optimierung von Computer-Modellen*, Tesis doctoral. Reimprimido por Birkhäuser (1977).

[23] BEASLEY, D.,BULL, D. R.,MARTIN, R. R. *(1993)*, *An overview of genetic algorithms: Part 1, fundamentals.*, University Computing, 15(2):58-69.

[24] HERNÁNDEZ, J. A., DORADO, J., GESTAL, M., PORTO, A. B. *(2005)*, *Avances en Algoritmos Evolutivos*, Universidad Nacional de Colombia (Sede Medellín) & Universidade da Coruña, 39-40.

[25] BONGARD, J. C. *(2013)*, *Evolutionary Robotics: Taking a biologically inspired approach to the design of autonomous, adaptive machines*, Communications of the ACM. 08/2013, VOL.56, NO.08.

[26] ABB®, *[Web en línea]*, <*http://www.abb.com*>, [Consulta: 09-03-2015]

[27] PARROT®, *[Web en línea]*, <*http://www.parrot.com/es*>, [Consulta: 09-03-2015]

[28] LONG, J. *(2012)*, *Darwin's Devices: What Evolving Robots Can Teach Us about the History of Life and the Future of Technology*, Basic Books.

[29] LUKE, S.,SPECTOR, L. *(1996)*, *Evolving teamwork and coordination with genetic programming*, Proceedings of the First Annual Conference on Genetic Programming. MIT Press, Cambridge, MA, 150–156.

[30] "SYMBRION" PROJECT, *[Web en línea]*, <*https://robotification.wordpress.com /2008/03/18/symbiotic-evolutionary-robot-organisms-project/*>, [Consulta: 09-03-2015]

[31] SHEN, W. M.,YIM, M.,SALEMI, B.,RUS, D.,MOLL, M.,LIPSON, H.,KLAVINS, E.,CHIRIKJIAN, G. S. *(2007)*, *Modular self-reconfigurable robot systems (grand challenges of robotics)*, Robotics & Automation Magazine 14, 1. IEEE, 43–52.

[32] MENG, Y.,ZHANG, Y.,JIN, Y *(2011)*, *Autonomous self-reconfiguration of modular robots by evolving a hierarchical mechanochemical model*, Computational Intelligence Magazine 6, 1. IEEE, 43–54.

[33] RIEFFEL, J.,SAUNDERS, F.,NADIMPALLI, S.,ZHOU, H.,HASSOUN, S.,RIFE, J.,TRIMMER, B. *(2009)*, *Evolving soft robotic locomotion in PhysX*, Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. ACM, NY, 2499–2504.

[34] GONZÁLEZ DE SANTOS, P.,GARCÍA, E.,ESTREMERA, J. *(2006)*, *Quadrupedal Locomotion: An Introduction to the Control of Four-legged Robots*, Instituto de Automática Industrial, CAR-CSIC(UPM), Springer, ISBN: 978-1-84628-306-2.

[35] WILLIAMS, D. M. *(2005)*, *Book of Insect Records*, University of Florida.

[36] BOSTON DYNAMICS®, *[Web en línea]*, <*http://www.bostondynamics.com*>, [Consulta: 09-03-2015]

[37] AIBO®, SONY, *[Web en línea]*, <*http://www.sony-aibo.com/*>, [Consulta: 10-03-2015]

[38] TROSSENROBOTICS, *[Web en línea]*, <*http://www.trossenrobotics.com/p/PhantomX-AX-12-Quadruped.aspx*>, [Consulta: 10-03-2015]

[39] CRUSTCRAWLER, *[Web en línea]*, <*http://www.crustcrawler.com/products/quadrod/index.php?prod=18*>, [Consulta: 10-03-2015]

[40] LYNXMOTION, *[Web en línea]*, <*http://www.lynxmotion.com/c-26-quadrapods.aspx*>, [Consulta: 10-03-2015]

[41] GESTAL, M., RIVERO, D., RABUÑAL, J. R., DORADO, J., PAZOS, A. *(2010)*, *Introducción a los Algoritmos Genéticos y la Programación Genética*, Universidade da Coruña, Servizo de Publicacións. ISBN: 978-84-9749-422-9

[42] BRINDLE, A. *(1981)*, *Genetic Algorithms for Function Optimization*, Ph.D. Thesis, University of Alberta.

[43] BOOKER, L. B. *(1982)*, *Intelligent Behaviour as an Adaptation to the Task Environment*, Ph.D. Dissertation, University of Michigan.

[44] BAKER, J. E. *(1987)*, *Reducing bias and inefficiency in the selection algorithm*, Proceedings of the Second International Conference of Genetic Algoritm, 14-21, Cambridge MA, Lawrence Erlbaum Associates.

[45] JONG, K. A. D. *(1975)*, *An analysis of the behaviour of a class of genetic adaptive systems*, Ph.D. Thesis, University of Michigan.

[46] HERRERA, F.,LOZANO, M.,VERDEGAY, J. L. *(1998)*, *Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis*, Artificial Intelligence Review 12, 265–319, Kluwer Academic Publishers.

[47] RADCLIFFE, N. J. *(1991)*, *Equivalence Class Analysis of Genetic Algorithms*, Complex Systems 5(2), 183–205.

[48] ESHELMAN, L. J.,SCHAFFER, J. D. *(1993)*, *Real-Coded Genetic Algorithms and Interval-Schemata*, Foundation of Genetic Algorithms 2, 187–202, L.D. Whitley (Ed.).

[49] MICHALEWICZ, Z. *(1992)*, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York.

[50] WRIGHT, A. *(1991)*, *Genetic Algorithms for Real Parameter Optimization*, Foundations of Genetic Algorithms 1, 205–218, G.J.E Rawlin (Ed.).

[51] MÜHLENBEIN, H.,SCHLIERKAMP-VOOSEN, D. *(1993)*, *Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization*, Evolutionary Computation 1, 25-49.

[52] WRIGHT, A. *(1990)*, *Genetic Algorithms for Real Parameter Optimization*, Foundations of Genetic Algorithms, First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, 205–218, G.J.E. Rawlin(Ed.).

[53] KOZA, J. R. *(1992)*, *Genetic Programming: On the programming of computers by means of natural selection*, MIT Press, 95-98, ISBN: 0-262-11170-5.

[54] V-REP®, *[Web en línea]*, <*http://www.coppeliarobotics.com/index.html*>, [Consulta: 22-04-2015]

[55] MANUAL DE USUARIO DE V-REP, *[Web en línea]*, <*http://www.coppeliarobotics.com/helpFiles/index.html*>, [Consulta: 22-04-2015]

[56] FORO DE V-REP, *[Web en línea]*, <*http://www.forum.coppeliarobotics.com/index.php*>, [Consulta: 22-04-2015]

[57] GONZÁLEZ, J.: FREECAD FUTABA 3003, *[Web en línea]*, <*http://www.iearobotics.com/wiki/index.php?title=Freecad:_Futaba_3003*>, [Consulta: 22-04-2015]

[58] RIGID BODY TUTORIAL, *[Web en línea]*, <*http://www.coppeliarobotics.com/helpFiles/en/rigidBodyTutorial.htm*>, [Consulta: 25-04-2015]

[59] COLLISION DETECTION, *[Web en línea]*, <*http://www.coppeliarobotics.com/helpFiles/en/collisionDetection.htm*>, [Consulta: 25-04-2015]

[60] MATLAB®, *[Web en línea]*, <*http://es.mathworks.com/products/matlab/*>, [Consulta: 26-04-2015]

[61] REMOTE API MODUS OPERANDI, *[Web en línea]*, <*http://www.coppeliarobotics.com/helpFiles/en/remoteApiModusOperandi.htm*>, [Consulta: 27-04-2015]

[62] REMOTE API FUNCTIONS (MATLAB), *[Web en línea]*, <*http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm*>, [Consulta: 27-04-2015]

[63] DE COS CASTILLO, M. *(1995)*, *Teoría General del Proyecto: Dirección de Proyectos*, Volumen I, Síntesis Editorial.

[64] DE COS CASTILLO, M. *(1997)*, *Teoría General del Proyecto: Ingeniería de Proyectos*, Volumen II, Síntesis Editorial.

8

Planificación y Presupuesto

En este capítulo se detallan aquellos aspectos relacionados con la gestión del proyecto. De este modo, se exponen la planificación del proyecto, la Estructura de Descomposición del Proyecto (EDP), el Diagrama de Gantt y el presupuesto necesario para completar este proyecto.

8.1. Planificación

La planificación del presente proyecto puede dividirse en cuatro grupos de tareas, cuyo cumplimiento se detalla a continuación:

1. **Dirección del Proyecto**: Consta de la planificación, del control del desarrollo del proyecto y de la redacción de la memoria del Trabajo Fin de Grado.
   - La **planificación** se realiza al comienzo del proyecto, estableciéndose como objetivo del proyecto la realización de un sistema de aprendizaje evolutivo para un robot caminante. Una vez tomada esta decisión, se procede a leer artículos acerca del tema.
   - El **control del desarrollo** y las **reuniones** se realizan durante todo el proyecto, proponiendo soluciones a los problemas que van apareciendo y comprobando la correcta realización del proyecto.
   - La **redacción de la memoria del Trabajo Fin de Grado** tiene lugar al finalizar el proyecto.
2. **Estudios previos y aprendizaje**: Con el objetivo del proyecto decidido, se procede a determinar las **condiciones** que debe satisfacer el sistema, en cuanto a la codificación de los individuos, la función de coste, la programación del algoritmo, como debe ser el robot, etc. Posteriormente, se estudian las **herramientas** que se utilizarán, que son V-REP, Matlab, Arduino y el robot cuadrúpedo; y se aprende a manejarlas.
3. **Diseño y Desarrollo del software**: Una vez se ha analizado el proyecto y se ha aprendido a manejar las herramientas, se procede a **diseñar el modelo de simulación** y a **programar el algoritmo**.
4. **Implementación y Ensayos**: Finalmente, tras comprobar que el entrono de simulación funciona correctamente y que el algoritmo se desarrolla de manera adecuada, se realizan las **simulaciones** pertinentes y los **experimentos con el robot real**.

La **Estructura de Descomposición del Proyecto** es una descomposición simple y organizada del trabajo requerido para completar un proyecto. En la Figura 8.1 se puede observar la **EDP** del presente proyecto [63] [64].

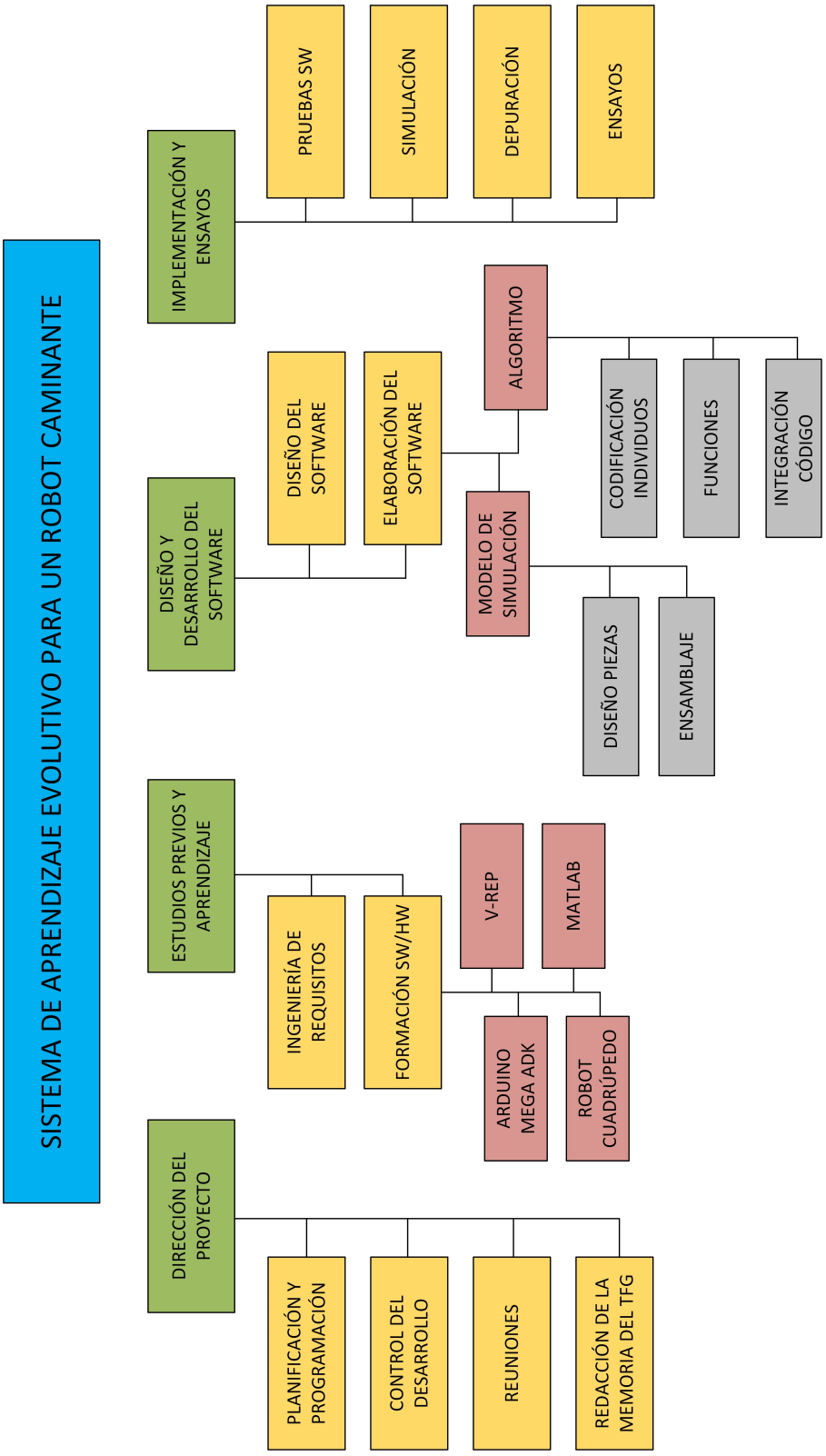


Figura 8.1: *Estructura de Descomposición del Proyecto*

El **Diagrama de Gantt** es una herramienta que se emplea en gestión de proyectos para representar las diferentes fases, tareas y actividades programadas como parte del proyecto. Permite exponer el tiempo de dedicación previsto para las fases, tareas y actividades del proyecto. El diagrama de Gantt está formado por "Unidades Mínimas de Trabajo", que son el elemento de trabajo más pequeño e indivisible. Se utilizan para planificar el proyecto [63] [64]. En la Figura 8.2 se muestra el **Diagrama de Gantt del Proyecto**.

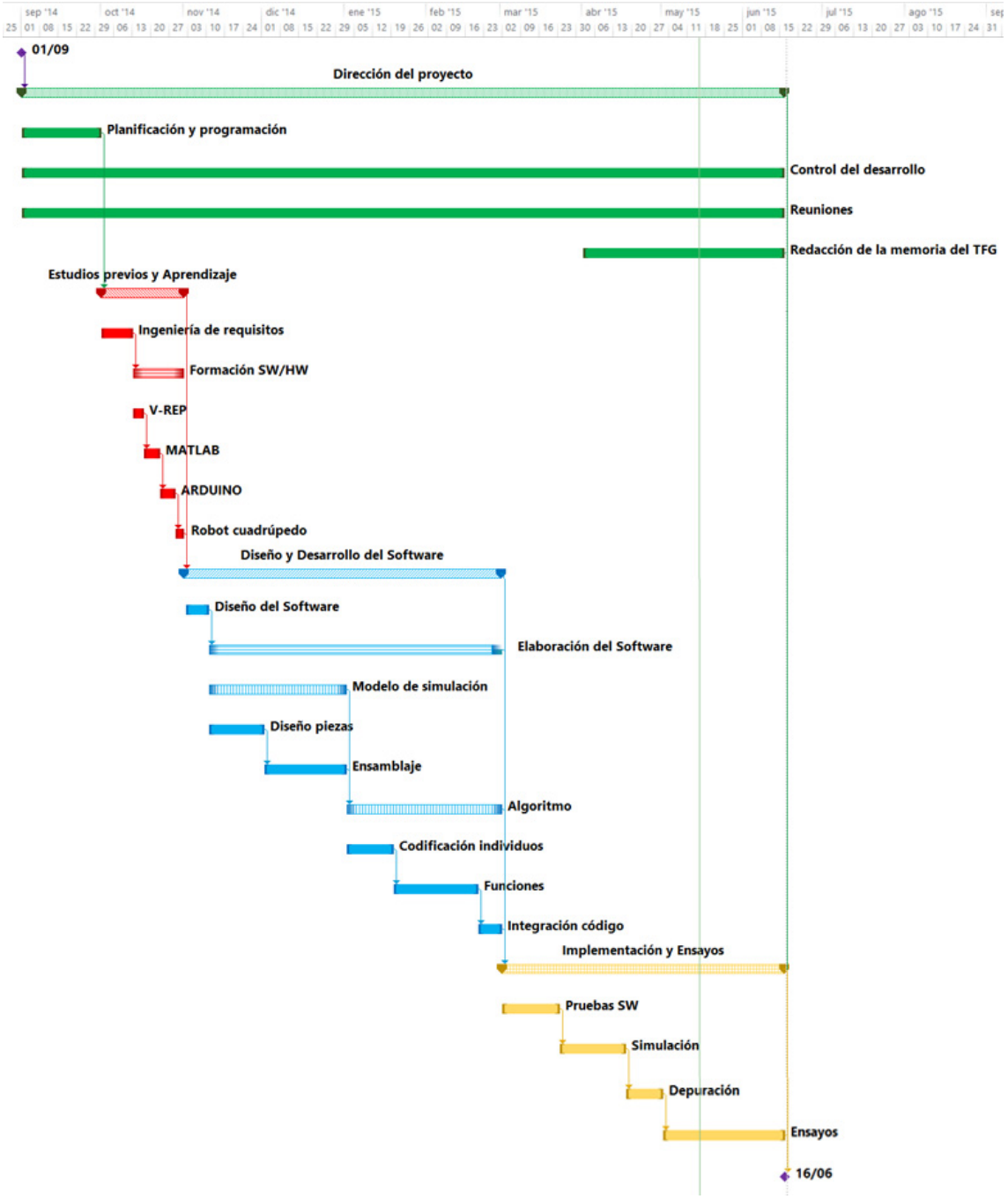


Figura 8.2: *Diagrama de Gantt del Proyecto*

8.2. Presupuesto

Debido al carácter de I+D+i del presente proyecto, el presupuesto debe cuantificar por un lado el coste del material y de los equipos adquiridos específicamente para el desarrollo del mismo y por otro lado una evaluación aproximada del coste de personal que dicha actividad ha conllevado. Cabe destacar que los costes de personal han sido cuantificados de una manera aproximada, obviando así una evaluación económica rigurosa por tratarse de personal perteneciente a la Universidad. De esta forma, los presentes presupuestos reflejan una aproximación del coste necesario para replicar el proyecto, sin haber sido necesario gastar lo indicado para la realización del mismo.

Los presupuestos se han organizado de la siguiente manera:

- Presupuesto total del proyecto.
- Presupuesto personal.
- Presupuesto material del robot.
- Presupuesto general de los equipos.

PRESUPUESTO TOTAL	**IMPORTE (€)**
Total personal	3020
Total material del robot	158,18
Total equipo	1340
TOTAL	4518,18
21 % IVA	948,8178
TOTAL	**5466,9978**

COSTE DE PERSONAL	**PRECIO UNITARIO (€/h)**	**Nº HORAS**	**IMPORTE (€)**
Alumno	4	380	1520
Tutor	30	50	1500
SUBTOTAL			**3020**

COSTE MATERIAL DEL ROBOT	**CANTIDAD**	**PRECIO (€)**	**IMPORTE (€)**
Arduino Mega ADK	1	58,98	58,98
Servo Futaba S3003	8	9	72
BEC entrada 7,4V salida 5-6V	1	12,2	12,2
Batería LiPo 2S 1800mAh	1	15	15
SUBTOTAL			**158,18**

EQUIPOS Y SOFTWARE	**CANTIDAD**	**PRECIO (€)**	**IMPORTE (€)**
Matlab (MathWorks ®)	1	500[1]	500
Ordenador	1	840[2]	840
SUBTOTAL			**1340**

[1]This license price applies for purchase and use in Spain. Likewise, applies for a faculty, staff, or researcher at a degree-granting institution. You may to operate, install, and administer the software yourself.

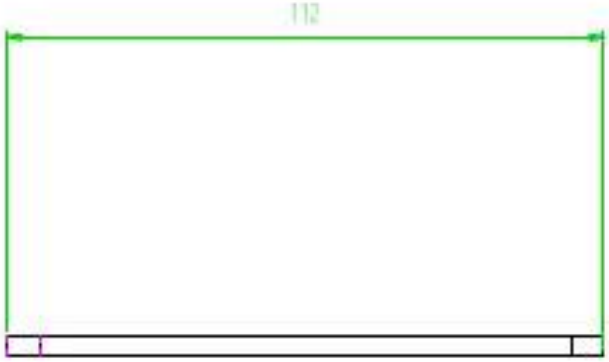
[2]Es necesario tener en cuenta que el ordenador portátil no es de uso exclusivo, pudiendo así amortizarlo con otros usos y durante más tiempo.

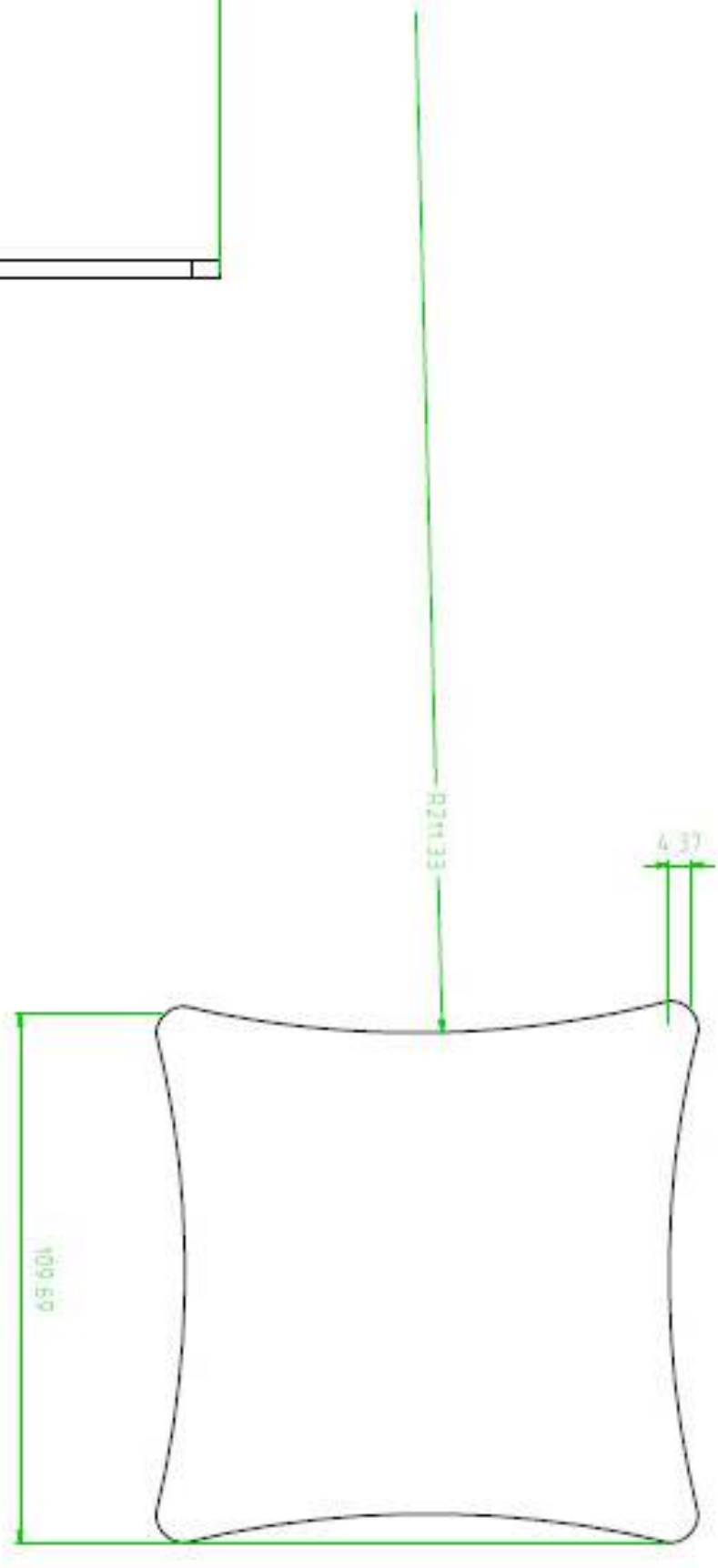
9

ANEXOS

9.1. ANEXO I - Planos del robot real

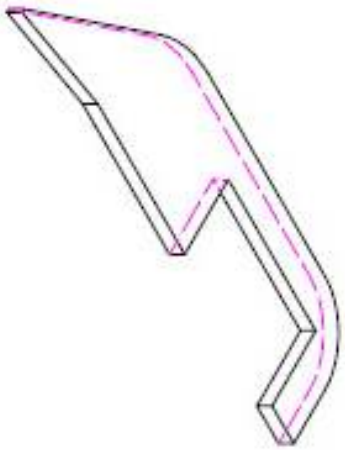
En este documento se incluyen los planos de las principales piezas del robot real, que han sido obtenidos de la referencia [11].

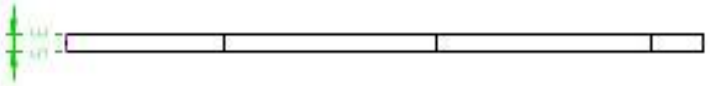


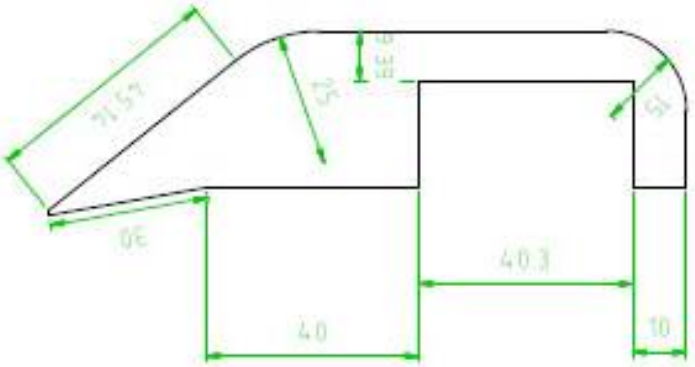


FILE NAME	
SIZE	
DRAWN	
CHKD	
APPVD	
ISSUED	
REV	
DIMENSIONS	
FINISH	
SHEET	
SCALE	

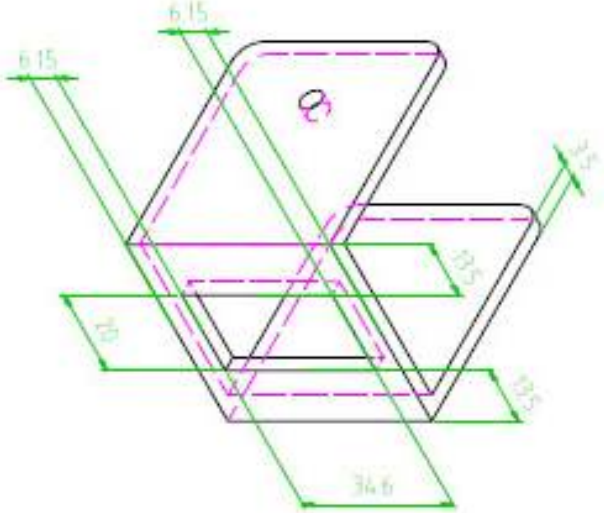
PRODUCED BY AN AUTODESK EDUCATIONAL PRODUCT

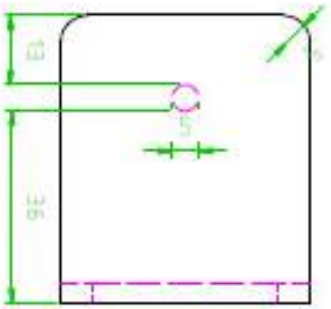


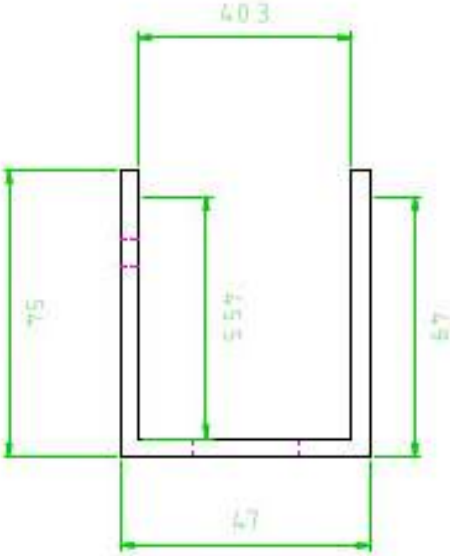




FILE NAME	
SIZE	
DRAWN	
CHECK	
APPR.	
ISSUED	
REV	
CONTRACT NO	
PITCH NO	
SHEET	
DWG NO	–
SCALE	







FILE NAME	
SIZE	PLOT NO
DRAWN	
CHECK	SHEET
APPR.	
ISSUED	SCALE
REV	
CONTRACT NO	DWG NO
	–

9.2. ANEXO II - Código para las pruebas del entorno de trabajo

```
function pruebas()
        %Iniciamos el programa y abrimos la conexion
        disp('Programa iniciado');
        vrep=remApi('remoteApi'); %Se crea el archivo servidor (utiliza
            remoteApiProto.m)
        vrep.simxFinish(-1); %Por si acaso, se cierran todas las conexiones
             previamente abiertas
        clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5); %Se
            establece el cliente de vrep

        if (clientID>-1)
                disp('Conectado al servidor API remoto correctamente');

                %Nombramos las juntas tal y como estan en la jerarquia de
                    vrep
                jointHandles=[-1 -1 -1 -1 -1 -1 -1 -1];
                aux=1;
                for i=1:4
                        for j=1:2
                                jointname=['Junta' int2str(j) '_' int2str(i
                                    )];
                                [err1,jointHandles(aux)]=vrep.
                                    simxGetObjectHandle(clientID, jointname,
                                     vrep.simx_opmode_oneshot_wait);
                                aux=aux+1;
                        end;
                end;

                [err1,BodyHandle]=vrep.simxGetObjectHandle(clientID, 'Body'
                    , vrep.simx_opmode_oneshot_wait); %Nombro el cuerpo del
                    robot tal y como esta en la jerarquIa de vrep

                pasos=0; %Inicializo en numero de pasos

                %Obtengo la posicion inicial "posi"
                [err4,posi]=vrep.simxGetObjectPosition(clientID, BodyHandle
                    , -1, vrep.simx_opmode_streaming); %Se inicia el
                    streaming de la posicion
                while true
                        [err4,posi]=vrep.simxGetObjectPosition(clientID,
                            BodyHandle, -1, vrep.simx_opmode_buffer); %Se
                            intenta obtener la posicion
```

```
        if(err4==vrep.simx_return_ok) %Asegurarse de que ha
            dado tiempo a que el primer dato llegue
                fprintf('Posicion incial x: %d\n',posi(1));
                fprintf('Posicion incial y: %d\n',posi(2));
                break
        end;
end;

while(pasos<10)
%PRIMER PASO CON LAS PATAS 1 Y 3%

%Levanto Junta 2 pata 1
vrep.simxSetJointTargetPosition(clientID, jointHandles(2),
    +20*pi/180, vrep.simx_opmode_oneshot);
[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (2), vrep.simx_opmode_streaming);
while true
    [err2,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(2), vrep.simx_opmode_buffer);
    if(abs(v-20*pi/180)<0.1*pi/180)
                break
    end;
end;

%Coloco Junta 1 pata 1
vrep.simxSetJointTargetPosition(clientID, jointHandles(1),
    -45*pi/180, vrep.simx_opmode_oneshot);
[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (1), vrep.simx_opmode_streaming);
while true
    [err2,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(1), vrep.simx_opmode_buffer);
    if(abs(v+45*pi/180)<0.1*pi/180)
                break
    end;
end;

%Bajo Junta 2 pata 1
vrep.simxSetJointTargetPosition(clientID, jointHandles(2),
    -0*pi/180, vrep.simx_opmode_oneshot);
[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (2), vrep.simx_opmode_streaming);
while true
    [err2,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(2), vrep.simx_opmode_buffer);
    if(abs(v+0*pi/180)<0.1*pi/180)
                break
    end;
end;

%Levanto Junta 2 pata 3
vrep.simxSetJointTargetPosition(clientID, jointHandles(6),
    +20*pi/180, vrep.simx_opmode_oneshot);
```

```
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (6), vrep.simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(6), vrep.simx_opmode_buffer);
    if(abs(v-20*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Coloco Junta 1 pata 3
vrep.simxSetJointTargetPosition(clientID, jointHandles(5),
    +45*pi/180, vrep.simx_opmode_oneshot);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (5), vrep.simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(5), vrep.simx_opmode_buffer);
    if(abs(v-45*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Bajo Junta 2 pata 3
vrep.simxSetJointTargetPosition(clientID, jointHandles(6),
    -0*pi/180, vrep.simx_opmode_oneshot);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (6), vrep.simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(6), vrep.simx_opmode_buffer);
    if(abs(v+0*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Avanzo Junta 1 patas 1 y 3
vrep.simxPauseCommunication(clientID, 1); %se pausa la
    conexion para poder mover varias juntas a la vez
vrep.simxSetJointTargetPosition(clientID, jointHandles(1),
    +0*pi/180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(5),
    -0*pi/180, vrep.simx_opmode_oneshot);
vrep.simxPauseCommunication(clientID, 0); %se reestablece
    la conexion

[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (1), vrep.simx_opmode_streaming);
while true
    [err2,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(1), vrep.simx_opmode_buffer);
    if(abs(v-0*pi/180)<0.1*pi/180)
                    break
```

```
    end;
end;

[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (5), vrep.simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(5), vrep.simx_opmode_buffer);
    if(abs(v+0*pi/180)<0.1*pi/180)
                    break
    end;
end;


%SEGUNDO PASO CON LAS PATAS 2 Y 4%

%Levanto Junta 2 pata 2
vrep.simxSetJointTargetPosition(clientID, jointHandles(4),
    +20*pi/180, vrep.simx_opmode_oneshot);
[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (4), vrep.simx_opmode_streaming);
while true
    [err2,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(4), vrep.simx_opmode_buffer);
    if(abs(v-20*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Coloco Junta 1 pata 2
vrep.simxSetJointTargetPosition(clientID, jointHandles(3),
    -45*pi/180, vrep.simx_opmode_oneshot);
[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (3), vrep.simx_opmode_streaming);
while true
    [err2,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(3), vrep.simx_opmode_buffer);
    if(abs(v+45*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Bajo Junta 2 pata 2
vrep.simxSetJointTargetPosition(clientID, jointHandles(4),
    -0*pi/180, vrep.simx_opmode_oneshot);
[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (4), vrep.simx_opmode_streaming);
while true
    [err2,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(4), vrep.simx_opmode_buffer);
    if(abs(v+0*pi/180)<0.1*pi/180)
                    break
    end;
```

```
end;

%Levanto Junta 2 pata 4
vrep.simxSetJointTargetPosition(clientID, jointHandles(8),
    +20*pi/180, vrep.simx_opmode_oneshot);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (8), vrep.simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(8), vrep.simx_opmode_buffer);
    if(abs(v-20*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Coloco Junta 1 pata 4
vrep.simxSetJointTargetPosition(clientID, jointHandles(7),
    +45*pi/180, vrep.simx_opmode_oneshot);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (7), vrep.simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(7), vrep.simx_opmode_buffer);
    if(abs(v-45*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Bajo Junta 2 pata 4
vrep.simxSetJointTargetPosition(clientID, jointHandles(8),
    -0*pi/180, vrep.simx_opmode_oneshot);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (8), vrep.simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(8), vrep.simx_opmode_buffer);
    if(abs(v+0*pi/180)<0.1*pi/180)
                    break
    end;
end;

%Avanzo Junta 1 patas 2 y 4
vrep.simxPauseCommunication(clientID, 1); %se pausa la
    conexion para poder mover varias juntas a la vez
vrep.simxSetJointTargetPosition(clientID, jointHandles(3),
    +0*pi/180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(7),
    -0*pi/180, vrep.simx_opmode_oneshot);
vrep.simxPauseCommunication(clientID, 0); %se reestablece
    la conexion

[err2,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (3), vrep.simx_opmode_streaming);
```

```
                while true
                    [err2,v]=vrep.simxGetJointPosition(clientID,
                        jointHandles(3), vrep.simx_opmode_buffer);
                    if(abs(v-0*pi/180)<0.1*pi/180)
                                break
                    end;
                end;

                [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
                    (7), vrep.simx_opmode_streaming);
                while true
                    [err3,v]=vrep.simxGetJointPosition(clientID,
                        jointHandles(7), vrep.simx_opmode_buffer);
                    if(abs(v+0*pi/180)<0.1*pi/180)
                                break
                    end;
                end;

                pasos=pasos+1;
                end;

                %Obtengo la posicion final "posf"
                [err4,posf]=vrep.simxGetObjectPosition(clientID, BodyHandle
                    , -1, vrep.simx_opmode_streaming); %Se inicia el
                    streaming de la posicion
                while true
                        [err4,posf]=vrep.simxGetObjectPosition(clientID,
                            BodyHandle, -1, vrep.simx_opmode_buffer); %Se
                            intenta obtener la posicion
                        if(err4==vrep.simx_return_ok) %Asegurarse de que ha
                             dado tiempo a que el primer dato llegue
                                fprintf('Posicion final x: %d\n',posf(1));
                                fprintf('Posicion final y: %d\n',posf(2));
                                break
                        end;
                end;

                %Obtengo el avance
                d=sqrt(((posf(1)-posi(1))^2)+((posf(2)-posi(2))^2));
                fprintf('El robot ha avanzado: %d\n',d);

                vrep.simxFinish(clientID); %cerramos la conexion
        else
                disp('Fallo al conectar con el servidor API remoto');
        end

        vrep.delete(); %Se llama al destructor
        disp('Program ended');

end
```

9.3. ANEXO III - Algoritmo Genético

```
function solucion=genetico(NumGeneraciones,NumIndividuos,LimMejora,pC,pM)
% genetico(NumGeneraciones,NumIndividuos,LimMejora,pC,pM)
% Calcula la solucion optima para maximizar el avance
% del robot mediante algoritmos geneticos.
% Emplea seleccion por ranking y cruce por un punto.

clc
LongInd=8*3; %8 motores con 3 parametros cada uno (amplitud, frecuencia y
    retraso de la senoide)
NumMotores=8; %el robot tiene 8 motores
NumMuestras=5; %numero de muestras que se quieran para el seno de cada
    motor
limite=0; %variable para controlar la convergencia
MejorFitAnt=0; %inicializar el mejor fitness anterior
NumPasos=4; %numero de pasos que dara el individuo mas apto al finalizar

poblacion=PoblacionInicial(NumIndividuos,NumMotores); %generamos la
    poblacion inicial
disp('Poblacion inicial creada')

for k=1:NumGeneraciones
        X=sprintf('Comienza la generacion numero %d',k);
        disp(X)
        disp(' ')

        poblacion=Cruce(poblacion,NumMotores,pC,pM); %se realiza el cruce y
            si fuera necesario la mutacion (devuelve el doble de individuos
            )

        [MejorFit,poblacion]=Seleccion(poblacion,NumMotores,NumMuestras); %
            se realiza la seleccion (se quitan la mitad de los peores
            individuos)

        %se utiliza el fitness del individuo mas apto para saber si, tras
            un numero de veces determinado, el individuo mas apto no ha
            mejorado
        %y, por tanto, la poblacion ha convergido
        if(MejorFit>MejorFitAnt) %he mejorado
                limite=0;
        else %no he mejorado
                limite=limite+1;
                if(limite==LimMejora) %el algoritmo ha convergido
                        %Solucion igual al mejor individuo
                        disp('La poblacion ha convergido')
```

```
                        disp('␣')
                        solucion=poblacion(1,:);

                        %Se mueve el robot con el individuo mas apto
                            durante algunos pasos
                        muestras=Traducir(solucion,NumMotores,NumMuestras);
                        MoverFinal(muestras,NumPasos);
                        return
                end;
        end;
        MejorFitAnt=MejorFit; %actualizo el mejor fitness anterior

        disp('␣')
        X=sprintf('La␣generacion␣numero␣%d␣ha␣finalizado',k);
        disp(X)
end;

%Solucion igual al mejor individuo
disp('El␣Algoritmo␣Genetico␣ha␣finalizado')
disp('␣')
solucion=poblacion(1,:);

%Se mueve el robot con el individuo mas apto durante algunos pasos
muestras=Traducir(solucion,NumMotores,NumMuestras);
MoverFinal(muestras,NumPasos);
```

```
function poblacion=PoblacionInicial(NumIndividuos,NumMotores)
%Esta funcion devuelve una matriz de tantas filas como individuos y 8*3
    columnas (tantas como parametros)
%Dicha matriz contiene la poblacion

for i=1:NumIndividuos

        %Generamos los valores aleatorios de los parametros de cada motor (
            hay 8 motores) y formamos un individuo
        for j=1:NumMotores
                motor=[45*rand (rand*(pi-(pi/3))+(pi/3)) 2*pi*rand];
                if(j==1)
                        individuo=[motor];
                else
                        individuo=[individuo motor];
                end;
        end;

        %Incluimos el individuo creado a la poblacion
        if(i==1)
                poblacion=[individuo];
        else
                poblacion=[poblacion; individuo];
        end;

end;
```

```
function PobCruzada=Cruce(individuos,NumMotores,pC,pM)
%Esta funcion devuelve una matriz de tantas filas como el doble de
    individuos y 8*3 columnas (tantas como parametros)
%Dicha matriz contiene la poblacion (hay que eliminar la peor mitad en la
    seleccion)

[NumInd,LongInd]=size(individuos); %se guarda el numero de individuos y la
    longitud de estos
n=NumInd;

while(n~=2*NumInd) %como en la seleccion eliminamos la mitad, debemos hacer
     cruces hasta que completemos el doble del numero inicial de individuos
        aleatorio_padre=ceil(rand*NumInd); %elijo una posicion aleatoria
            para el padre
        aleatorio_madre=ceil(rand*NumInd); %elijo una posicion aleatoria
            para la madre
        aleatorio_cruce=rand; %lo usaremos para saber si cruzamos
        aleatorio_mutacion=rand; %lo usaremos para saber si mutamos

    while(aleatorio_padre==aleatorio_madre) %si el padre y la madre son el
        mismo individuo, cambiamos a la madre
        aleatorio_madre=ceil(rand*NumInd);
    end;

    padre=individuos(aleatorio_padre,:); %seleccionamos el padre
    madre=individuos(aleatorio_madre,:); %seleccionamos la madre

        if(pC>aleatorio_cruce) %cruzamos si es necesario
                aleatorio_tipocruce=ceil(2*rand);
                if(aleatorio_tipocruce==1) %cruce en un punto (la mitad)
                        hijo=[padre(1:floor(LongInd/2)) madre(floor(LongInd
                            /2)+1:end)];
                else
                        hijo=[madre(1:floor(LongInd/2)) padre(floor(LongInd
                            /2)+1:end)];
                end;

                if(pM>aleatorio_mutacion) %mutamos si es necesario
                        mascara=round(rand(1,length(hijo))); %genero una
                            mascara aleatoria con la que sabre que
                            parametros mutar
                        %con este bucle genero un individuo referencia para
                             mutar el hijo
                        for j=1:NumMotores
                                motor=[45*rand (rand*(pi-(pi/3))+(pi/3)) 2*
                                    pi*rand];
                                if(j==1)
                                        mutacion=[motor];
                                else
                                        mutacion=[mutacion motor];
                                end;
```

```
                    end;
                %dos parametros que tengan un 1 en la mascara se
                    sustituyen por la media entre el mismo y el valor de
                    dicho parametro en el individuo mutacion
                    for i=1:(length(hijo))
                            if((mascara(i))==1)
                                    hijo(i)=((hijo(i)+mutacion(i))/2);
                            end;
                    end;
            end; %fin mutacion

        else %si no hay cruce copiamos uno de los padres en el hijo
                aleatorio_copia=ceil(2*rand);

                if(aleatorio_copia==1)
                        hijo=padre;
                else
                        hijo=madre;
                end;
        end;%fin cruce

        individuos=[individuos; hijo]; %se incluye el hijo a la poblacion
        n=n+1;
end;

PobCruzada=individuos; %guardamos el resultado
```

```
function [MejorFit,NuevaPoblacion]=Seleccion(poblacion,NumMotores,
    NumMuestras)
%Esta funcion devuelve una matriz de tantas filas como individuos y 8*3
    columnas (tantas como parametros)
%Dicha matriz contiene la poblacion que pasa a la siguiente generacion,
    tras haber eliminado a la peor mitad.
%Tambien devuelve el fitness del individuo mas apto para la posterior
    evaluacion de la convergencia

[NumInd,LongInd]=size(poblacion); %se guarda el numero de individuos y la
    longitud de estos

Fitness=EvaluarFitness(poblacion,NumMotores,NumMuestras); %se calcula el
    fitness de cada individuo

%Seleccion por Ranking
%En Y se ordenan LOS FITNESS de manera que los primeros sean los mejores (
    los que avancen mas)
%En I se ordenan LAS POSICIONES de los individuos segun sus fitness
%El vector Fitness no se ve modificado
[Y,I]=sort(Fitness,'descend');

NuevaPoblacion=poblacion(I(1:floor(length(I)/2)),:); %se escoge la mejor
    mitad para la siguiente generacion
MejorFit=Y(1); %se guarda el mejor fitness
```

```
function Fitness=EvaluarFitness(poblacion,NumMotores,NumMuestras)
%Esta funcion devuelve un vector fila en el que cada elemento es el fitness
    (avance del robot) de cada individuo
%con el objetivo de despues ordenarlo en la funcion Seleccion y asi escoger
    a la mejor mitad

[NumInd,LongInd]=size(poblacion); %se guarda el numero de individuos y la
   longitud de estos

for k=1:NumInd
        individuo=poblacion(k,:); %se selecciona un individuo para probarlo
            en el robot y obtener su fitness

        muestras=Traducir(individuo,NumMotores,NumMuestras); %se traduce y
           muestrea el individuo

        [posi,posf]=Mover(muestras); %movemos el robot en vrep con ese
           individuo muestreado y nos devuelve las posiciones inicial y
           final

        FitInd=sqrt(((posf(1)-posi(1))^2)+((posf(2)-posi(2))^2)); %
           calculamos el avance (su fitness)

        %Se incluye el fitness de ese individuo al vector de Fitness
        if(k==1)
                Fitness=[FitInd];
        else
                Fitness=[Fitness FitInd];
        end;
end;
```

```
function muestras=Traducir(individuo,NumMotores,NumMuestras)
%Esta funcion genera una tabla llamada muestras donde estan muestreados
   todos los senos de los motores,
%que quedan definidos segun los parametros del individuo que se quiere
   muestrear

np=1;
for j=1:NumMotores
        %Obtenemos del individuo A, w y fi, para ese motor
        A=individuo(np);
        w=individuo(np+1);
        fi=individuo(np+2);
        np=np+3;

        %Muestreamos el seno definido con los valores obtenidos
        for i=1:NumMuestras
                t=((i*2*pi)/NumMuestras);
                valor=A*sin((w*t)+fi);
                if(i==1)
```

```
                        VectVal=[valor];
                else
                        VectVal=[VectVal valor];
                end;
        end;

        %Incluimos la fila de ese motor a la tabla de muestras
        if(j==1)
                muestras=[VectVal];
        else
                muestras=[muestras; VectVal];
        end;
end;
```

```
function [posi,posf]=Mover(muestras)
%Esta funcion mueve el robot con un individuo previamente muestreado y
    devuelve las posiciones inicial y final

[NumMotores,NumMuestras]=size(muestras); %se guarda el numero de motores y
    el de muestras

vrep=remApi('remoteApi'); %usamos el archivo prototipo (remoteApiProto.m)
vrep.simxFinish(-1); %por si acaso, cerramos todas las conexiones
    previamente abiertas
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5); %establecemos
    el cliente de vrep
if (clientID>-1)
        %Nombramos las juntas y el cuerpo tal y como se llaman en la
            jerarquia de vrep
        jointHandles=[-1 -1 -1 -1 -1 -1 -1 -1];
        aux=1;
        for i=1:4
                for j=1:2
                        jointname=['Junta' int2str(j) '_' int2str(i)];
                        [err1,jointHandles(aux)]=vrep.simxGetObjectHandle(
                            clientID, jointname, vrep.
                            simx_opmode_oneshot_wait);
                        aux=aux+1;
                end;
        end;
        [err1,BodyHandle]=vrep.simxGetObjectHandle(clientID, 'Body', vrep.
            simx_opmode_oneshot_wait);

        %Nombramos a los objetos de colision tal y como se llaman en el
            modulo de colision de vrep
        collisionHandles=[-1 -1 -1 -1 -1 -1 -1 -1];
        [err1,collisionHandles(1)]=vrep.simxGetCollisionHandle(clientID, '
            Pata1ConBody', vrep.simx_opmode_oneshot_wait);
        [err1,collisionHandles(2)]=vrep.simxGetCollisionHandle(clientID, '
            Pata2ConBody', vrep.simx_opmode_oneshot_wait);
        [err1,collisionHandles(3)]=vrep.simxGetCollisionHandle(clientID, '
            Pata3ConBody', vrep.simx_opmode_oneshot_wait);
```

```
[err1,collisionHandles(4)]=vrep.simxGetCollisionHandle(clientID,'
    Pata4ConBody',vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(5)]=vrep.simxGetCollisionHandle(clientID,'
    Pata1ConPata2',vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(6)]=vrep.simxGetCollisionHandle(clientID,'
    Pata2ConPata3',vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(7)]=vrep.simxGetCollisionHandle(clientID,'
    Pata3ConPata4',vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(8)]=vrep.simxGetCollisionHandle(clientID,'
    Pata4ConPata1',vrep.simx_opmode_oneshot_wait);

%Obtengo la posicion inicial "posi"
[err2,posi]=vrep.simxGetObjectPosition(clientID,BodyHandle,-1,
    vrep.simx_opmode_streaming); %Se inicia el streaming de la
    posicion
while true
        [err2,posi]=vrep.simxGetObjectPosition(clientID,BodyHandle
            ,-1,vrep.simx_opmode_buffer); %Se intenta obtener la
            posicion
        if(err2==vrep.simx_return_ok) %Asegurarse de que ha dado
            tiempo a que el primer dato llegue
                break
        end;
end;

%Utilizo el individuo muestreado en muestras para mover el robot
for NMuest=1:NumMuestras
        vrep.simxPauseCommunication(clientID,1); %Todas las
            ordenes se mandan a la vez
        vrep.simxSetJointTargetPosition(clientID,jointHandles(1),
            muestras(1,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID,jointHandles(2),
            muestras(2,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID,jointHandles(3),
            muestras(3,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID,jointHandles(4),
            muestras(4,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID,jointHandles(5),
            muestras(5,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID,jointHandles(6),
            muestras(6,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID,jointHandles(7),
            muestras(7,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID,jointHandles(8),
            muestras(8,NMuest)*pi/180,vrep.simx_opmode_oneshot);
        vrep.simxPauseCommunication(clientID,0);
        %Muevo el motor 1
        t=clock;
        startTime=t(6);
        currentTime=t(6);
        [err3,v]=vrep.simxGetJointPosition(clientID,jointHandles
            (1),vrep.simx_opmode_streaming);
```

```
[err4,collisionState5]=vrep.simxReadCollision(clientID,
    collisionHandles(5), vrep.simx_opmode_streaming);
[err5,collisionState8]=vrep.simxReadCollision(clientID,
    collisionHandles(8), vrep.simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(1), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState5]=vrep.simxReadCollision(
                clientID, collisionHandles(5), vrep.
                simx_opmode_buffer);
            [err5,collisionState8]=vrep.simxReadCollision(
                clientID, collisionHandles(8), vrep.
                simx_opmode_buffer);
            if((err4==vrep.simx_return_ok) & (err5==vrep.
                simx_return_ok))
                    if((abs(v-(muestras(1,NMuest)*pi/180))
                        <0.1*pi/180) | (collisionState5==true
                        ) | (collisionState8==true))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 2
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (2), vrep.simx_opmode_streaming);
[err4,collisionState1]=vrep.simxReadCollision(clientID,
    collisionHandles(1), vrep.simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(2), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState1]=vrep.simxReadCollision(
                clientID, collisionHandles(1), vrep.
                simx_opmode_buffer);
            if(err4==vrep.simx_return_ok)
                    if((abs(v-(muestras(2,NMuest)*pi/180))
                        <0.1*pi/180) | (collisionState1==true
                        ))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 3
```

```
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (3), vrep.simx_opmode_streaming);
[err4,collisionState5]=vrep.simxReadCollision(clientID,
    collisionHandles(5), vrep.simx_opmode_streaming);
[err5,collisionState6]=vrep.simxReadCollision(clientID,
    collisionHandles(6), vrep.simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(3), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState5]=vrep.simxReadCollision(
                clientID, collisionHandles(5), vrep.
                simx_opmode_buffer);
            [err5,collisionState6]=vrep.simxReadCollision(
                clientID, collisionHandles(6), vrep.
                simx_opmode_buffer);
            if((err4==vrep.simx_return_ok) & (err5==vrep.
                simx_return_ok))
                    if((abs(v-(muestras(3,NMuest)*pi/180))
                        <0.1*pi/180) | (collisionState5==true
                        ) | (collisionState6==true))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 4
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (4), vrep.simx_opmode_streaming);
[err4,collisionState2]=vrep.simxReadCollision(clientID,
    collisionHandles(2), vrep.simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(4), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState2]=vrep.simxReadCollision(
                clientID, collisionHandles(2), vrep.
                simx_opmode_buffer);
            if(err4==vrep.simx_return_ok)
                    if((abs(v-(muestras(4,NMuest)*pi/180))
                        <0.1*pi/180) | (collisionState2==true
                        ))
                                break
                    end;
            end;
```

```
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 5
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (5), vrep.simx_opmode_streaming);
[err4,collisionState6]=vrep.simxReadCollision(clientID,
    collisionHandles(6), vrep.simx_opmode_streaming);
[err5,collisionState7]=vrep.simxReadCollision(clientID,
    collisionHandles(7), vrep.simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(5), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState6]=vrep.simxReadCollision(
                clientID, collisionHandles(6), vrep.
                simx_opmode_buffer);
            [err5,collisionState7]=vrep.simxReadCollision(
                clientID, collisionHandles(7), vrep.
                simx_opmode_buffer);
            if((err4==vrep.simx_return_ok) & (err5==vrep.
                simx_return_ok))
                    if((abs(v-(muestras(5,NMuest)*pi/180))
                        <0.1*pi/180) | (collisionState6==true
                        ) | (collisionState7==true))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 6
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
    (6), vrep.simx_opmode_streaming);
[err4,collisionState3]=vrep.simxReadCollision(clientID,
    collisionHandles(3), vrep.simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(6), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState3]=vrep.simxReadCollision(
                clientID, collisionHandles(3), vrep.
                simx_opmode_buffer);
            if(err4==vrep.simx_return_ok)
```

```
                    if ((abs(v-(muestras(6,NMuest)*pi/180))
                        <0.1*pi/180) | (collisionState3==true
                        ))
                            break
                    end;
            end;
        end;
        t=clock;
        currentTime=t(6);
    end;
    %Muevo el motor 7
    t=clock;
    startTime=t(6);
    currentTime=t(6);
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
        (7), vrep.simx_opmode_streaming);
    [err4,collisionState7]=vrep.simxReadCollision(clientID,
        collisionHandles(7), vrep.simx_opmode_streaming);
    [err5,collisionState8]=vrep.simxReadCollision(clientID,
        collisionHandles(8), vrep.simx_opmode_streaming);
    while ((currentTime-startTime)<0.5)
        [err3,v]=vrep.simxGetJointPosition(clientID,
            jointHandles(7), vrep.simx_opmode_buffer);
        if(err3==vrep.simx_return_ok)
                [err4,collisionState7]=vrep.simxReadCollision(
                    clientID, collisionHandles(7), vrep.
                    simx_opmode_buffer);
                [err5,collisionState8]=vrep.simxReadCollision(
                    clientID, collisionHandles(8), vrep.
                    simx_opmode_buffer);
                if((err4==vrep.simx_return_ok) & (err5==vrep.
                    simx_return_ok))
                    if((abs(v-(muestras(7,NMuest)*pi/180))
                        <0.1*pi/180) | (collisionState7==true
                        ) | (collisionState8==true))
                            break
                    end;
                end;
        end;
        t=clock;
        currentTime=t(6);
    end;
    %Muevo el motor 8
    t=clock;
    startTime=t(6);
    currentTime=t(6);
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles
        (8), vrep.simx_opmode_streaming);
    [err4,collisionState4]=vrep.simxReadCollision(clientID,
        collisionHandles(4), vrep.simx_opmode_streaming);
    while ((currentTime-startTime)<0.5)
        [err3,v]=vrep.simxGetJointPosition(clientID,
            jointHandles(8), vrep.simx_opmode_buffer);
```

```
            if(err3==vrep.simx_return_ok)
                    [err4,collisionState4]=vrep.simxReadCollision(
                        clientID, collisionHandles(4), vrep.
                        simx_opmode_buffer);
                    if(err4==vrep.simx_return_ok)
                            if((abs(v-(muestras(8,NMuest)*pi/180))
                                <0.1*pi/180) | (collisionState4==true
                                ))
                                        break
                            end;
                    end;
            end;
            t=clock;
            currentTime=t(6);
        end;

end;

%Obtengo la posicion final "posf"
[err2,posf]=vrep.simxGetObjectPosition(clientID, BodyHandle, -1,
    vrep.simx_opmode_streaming); %Se inicia el streaming de la
    posicion
while true
        [err2,posf]=vrep.simxGetObjectPosition(clientID, BodyHandle
            , -1, vrep.simx_opmode_buffer); %Se intenta obtener la
            posicion
        if(err2==vrep.simx_return_ok) %Asegurarse de que ha dado
            tiempo a que el primer dato llegue
                break
        end;
end;

%Tras mover el robot con el individuo, reinicializo los motores
    para poder probar despues otro individuo
vrep.simxPauseCommunication(clientID, 1); %Todas las ordenes se
    mandan a la vez
vrep.simxSetJointTargetPosition(clientID, jointHandles(1), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(2), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(3), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(4), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(5), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(6), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(7), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID, jointHandles(8), 0*pi
    /180, vrep.simx_opmode_oneshot);
vrep.simxPauseCommunication(clientID, 0);
```

```
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(1), vrep.
    simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(1),
        vrep.simx_opmode_buffer);
    if(abs(v-0*pi/180)<0.1*pi/180)
                    break
    end;
end;
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(2), vrep.
    simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(2),
        vrep.simx_opmode_buffer);
    if(abs(v-0*pi/180)<0.1*pi/180)
                    break
    end;
end;
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(3), vrep.
    simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(3),
        vrep.simx_opmode_buffer);
    if(abs(v-0*pi/180)<0.1*pi/180)
                    break
    end;
end;
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(4), vrep.
    simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(4),
        vrep.simx_opmode_buffer);
    if(abs(v-0*pi/180)<0.1*pi/180)
                    break
    end;
end;
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(5), vrep.
    simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(5),
        vrep.simx_opmode_buffer);
    if(abs(v-0*pi/180)<0.1*pi/180)
                    break
    end;
end;
[err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(6), vrep.
    simx_opmode_streaming);
while true
    [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(6),
        vrep.simx_opmode_buffer);
    if(abs(v-0*pi/180)<0.1*pi/180)
                    break
    end;
```

```
        end;
        [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(7), vrep.
            simx_opmode_streaming);
        while true
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(7),
                vrep.simx_opmode_buffer);
            if(abs(v-0*pi/180)<0.1*pi/180)
                        break
            end;
        end;
        [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(8), vrep.
            simx_opmode_streaming);
        while true
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(8),
                vrep.simx_opmode_buffer);
            if(abs(v-0*pi/180)<0.1*pi/180)
                        break
            end;
        end;

        vrep.simxFinish(clientID); %cerramos la conexion
else
        disp('Fallo al conectar con el servidor API remoto');
        %En caso de fallo al conectar se devuelven unas posiciones nulas y
            por tanto el avance sera cero
        posi=[0 0];
        posf=[0 0];
end;
vrep.delete(); %llamamos al destructor
```

```
function MoverFinal(muestras,NumPasos)
%Esta funcion mueve el robot con el mejor individuo resultante del
    algoritmo genetico durante un numero de pasos determinado

[NumMotores,NumMuestras]=size(muestras); %se guarda el numero de motores y
    el de muestras
pasos=0;

vrep=remApi('remoteApi'); %usamos el archivo prototipo (remoteApiProto.m)
vrep.simxFinish(-1); %por si acaso, cerramos todas las conexiones
    previamente abiertas
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5); %establecemos
    el cliente de vrep
if (clientID>-1)
        %Nombramos las juntas y el cuerpo tal y como se llaman en la
            jerarquia de vrep
        jointHandles=[-1 -1 -1 -1 -1 -1 -1 -1];
        aux=1;
        for i=1:4
                for j=1:2
                        jointname=['Junta' int2str(j) '_' int2str(i)];
```

```
                [err1,jointHandles(aux)]=vrep.simxGetObjectHandle(
                    clientID, jointname, vrep.
                    simx_opmode_oneshot_wait);
                aux=aux+1;
        end;
end;
[err1,BodyHandle]=vrep.simxGetObjectHandle(clientID, 'Body', vrep.
    simx_opmode_oneshot_wait);

%Nombramos a los objetos de colision tal y como se llaman en el
    modulo de colision de vrep
collisionHandles=[-1 -1 -1 -1 -1 -1 -1 -1];
[err1,collisionHandles(1)]=vrep.simxGetCollisionHandle(clientID, '
    Pata1ConBody', vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(2)]=vrep.simxGetCollisionHandle(clientID, '
    Pata2ConBody', vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(3)]=vrep.simxGetCollisionHandle(clientID, '
    Pata3ConBody', vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(4)]=vrep.simxGetCollisionHandle(clientID, '
    Pata4ConBody', vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(5)]=vrep.simxGetCollisionHandle(clientID, '
    Pata1ConPata2', vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(6)]=vrep.simxGetCollisionHandle(clientID, '
    Pata2ConPata3', vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(7)]=vrep.simxGetCollisionHandle(clientID, '
    Pata3ConPata4', vrep.simx_opmode_oneshot_wait);
[err1,collisionHandles(8)]=vrep.simxGetCollisionHandle(clientID, '
    Pata4ConPata1', vrep.simx_opmode_oneshot_wait);

while(pasos<NumPasos)
        %Utilizo el individuo muestreado en muestras para mover el
            robot
        for NMuest=1:NumMuestras
                vrep.simxPauseCommunication(clientID, 1); %Todas
                    las ordenes se mandan a la vez
                vrep.simxSetJointTargetPosition(clientID,
                    jointHandles(1), muestras(1,NMuest)*pi/180, vrep
                    .simx_opmode_oneshot);
                vrep.simxSetJointTargetPosition(clientID,
                    jointHandles(2), muestras(2,NMuest)*pi/180, vrep
                    .simx_opmode_oneshot);
                vrep.simxSetJointTargetPosition(clientID,
                    jointHandles(3), muestras(3,NMuest)*pi/180, vrep
                    .simx_opmode_oneshot);
                vrep.simxSetJointTargetPosition(clientID,
                    jointHandles(4), muestras(4,NMuest)*pi/180, vrep
                    .simx_opmode_oneshot);
                vrep.simxSetJointTargetPosition(clientID,
                    jointHandles(5), muestras(5,NMuest)*pi/180, vrep
                    .simx_opmode_oneshot);
                vrep.simxSetJointTargetPosition(clientID,
                    jointHandles(6), muestras(6,NMuest)*pi/180, vrep
                    .simx_opmode_oneshot);
```

```
vrep.simxSetJointTargetPosition(clientID,
    jointHandles(7), muestras(7,NMuest)*pi/180, vrep
    .simx_opmode_oneshot);
vrep.simxSetJointTargetPosition(clientID,
    jointHandles(8), muestras(8,NMuest)*pi/180, vrep
    .simx_opmode_oneshot);
vrep.simxPauseCommunication(clientID, 0);
%Muevo el motor 1
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(1), vrep.simx_opmode_streaming);
[err4,collisionState5]=vrep.simxReadCollision(
    clientID, collisionHandles(5), vrep.
    simx_opmode_streaming);
[err5,collisionState8]=vrep.simxReadCollision(
    clientID, collisionHandles(8), vrep.
    simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(1), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState5]=vrep.
                simxReadCollision(clientID,
                collisionHandles(5), vrep.
                simx_opmode_buffer);
            [err5,collisionState8]=vrep.
                simxReadCollision(clientID,
                collisionHandles(8), vrep.
                simx_opmode_buffer);
            if((err4==vrep.simx_return_ok) & (err5==
                vrep.simx_return_ok))
                    if((abs(v-(muestras(1,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState5==true) | (
                        collisionState8==true))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 2
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(2), vrep.simx_opmode_streaming);
[err4,collisionState1]=vrep.simxReadCollision(
    clientID, collisionHandles(1), vrep.
    simx_opmode_streaming);
```

```
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(2), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState1]=vrep.
                simxReadCollision(clientID,
                collisionHandles(1), vrep.
                simx_opmode_buffer);
            if(err4==vrep.simx_return_ok)
                    if((abs(v-(muestras(2,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState1==true))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 3
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(3), vrep.simx_opmode_streaming);
[err4,collisionState5]=vrep.simxReadCollision(
    clientID, collisionHandles(5), vrep.
    simx_opmode_streaming);
[err5,collisionState6]=vrep.simxReadCollision(
    clientID, collisionHandles(6), vrep.
    simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(3), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState5]=vrep.
                simxReadCollision(clientID,
                collisionHandles(5), vrep.
                simx_opmode_buffer);
            [err5,collisionState6]=vrep.
                simxReadCollision(clientID,
                collisionHandles(6), vrep.
                simx_opmode_buffer);
            if((err4==vrep.simx_return_ok) & (err5==
                vrep.simx_return_ok))
                    if((abs(v-(muestras(3,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState5==true) | (
                        collisionState6==true))
                                break
                    end;
            end;
    end;
```

```
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 4
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(4),vrep.simx_opmode_streaming);
[err4,collisionState2]=vrep.simxReadCollision(
    clientID,collisionHandles(2),vrep.
    simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(4),vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState2]=vrep.
                simxReadCollision(clientID,
                collisionHandles(2),vrep.
                simx_opmode_buffer);
            if(err4==vrep.simx_return_ok)
                    if((abs(v-(muestras(4,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState2==true))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 5
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(5),vrep.simx_opmode_streaming);
[err4,collisionState6]=vrep.simxReadCollision(
    clientID,collisionHandles(6),vrep.
    simx_opmode_streaming);
[err5,collisionState7]=vrep.simxReadCollision(
    clientID,collisionHandles(7),vrep.
    simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(5),vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState6]=vrep.
                simxReadCollision(clientID,
                collisionHandles(6),vrep.
                simx_opmode_buffer);
            [err5,collisionState7]=vrep.
                simxReadCollision(clientID,
```

```
                    collisionHandles(7), vrep.
                    simx_opmode_buffer);
                if((err4==vrep.simx_return_ok) & (err5==
                    vrep.simx_return_ok))
                    if((abs(v-(muestras(5,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState6==true) | (
                        collisionState7==true))
                            break
                    end;
                end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 6
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(6), vrep.simx_opmode_streaming);
[err4,collisionState3]=vrep.simxReadCollision(
    clientID, collisionHandles(3), vrep.
    simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(6), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState3]=vrep.
                simxReadCollision(clientID,
                collisionHandles(3), vrep.
                simx_opmode_buffer);
            if(err4==vrep.simx_return_ok)
                    if((abs(v-(muestras(6,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState3==true))
                            break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 7
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(7), vrep.simx_opmode_streaming);
[err4,collisionState7]=vrep.simxReadCollision(
    clientID, collisionHandles(7), vrep.
    simx_opmode_streaming);
```

```
[err5,collisionState8]=vrep.simxReadCollision(
    clientID, collisionHandles(8), vrep.
    simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(7), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState7]=vrep.
                simxReadCollision(clientID,
                collisionHandles(7), vrep.
                simx_opmode_buffer);
            [err5,collisionState8]=vrep.
                simxReadCollision(clientID,
                collisionHandles(8), vrep.
                simx_opmode_buffer);
            if((err4==vrep.simx_return_ok) & (err5==
                vrep.simx_return_ok))
                    if((abs(v-(muestras(7,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState7==true) | (
                        collisionState8==true))
                                break
                    end;
            end;
    end;
    t=clock;
    currentTime=t(6);
end;
%Muevo el motor 8
t=clock;
startTime=t(6);
currentTime=t(6);
[err3,v]=vrep.simxGetJointPosition(clientID,
    jointHandles(8), vrep.simx_opmode_streaming);
[err4,collisionState4]=vrep.simxReadCollision(
    clientID, collisionHandles(4), vrep.
    simx_opmode_streaming);
while ((currentTime-startTime)<0.5)
    [err3,v]=vrep.simxGetJointPosition(clientID,
        jointHandles(8), vrep.simx_opmode_buffer);
    if(err3==vrep.simx_return_ok)
            [err4,collisionState4]=vrep.
                simxReadCollision(clientID,
                collisionHandles(4), vrep.
                simx_opmode_buffer);
            if(err4==vrep.simx_return_ok)
                    if((abs(v-(muestras(8,NMuest)*pi
                        /180))<0.1*pi/180) | (
                        collisionState4==true))
                                break
                    end;
            end;
    end;
```

```
                    t=clock;
                    currentTime=t(6);
                end;

            end;

            pasos=pasos+1;
        end;

        %Reinicializo los motores
        vrep.simxPauseCommunication(clientID, 1); %Todas las ordenes se
            mandan a la vez
        vrep.simxSetJointTargetPosition(clientID, jointHandles(1), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID, jointHandles(2), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID, jointHandles(3), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID, jointHandles(4), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID, jointHandles(5), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID, jointHandles(6), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID, jointHandles(7), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxSetJointTargetPosition(clientID, jointHandles(8), 0*pi
            /180, vrep.simx_opmode_oneshot);
        vrep.simxPauseCommunication(clientID, 0);
        [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(1), vrep.
            simx_opmode_streaming);
        while true
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(1),
                vrep.simx_opmode_buffer);
            if(abs(v-0*pi/180)<0.1*pi/180)
                        break
            end;
        end;
        [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(2), vrep.
            simx_opmode_streaming);
        while true
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(2),
                vrep.simx_opmode_buffer);
            if(abs(v-0*pi/180)<0.1*pi/180)
                        break
            end;
        end;
        [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(3), vrep.
            simx_opmode_streaming);
        while true
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(3),
                vrep.simx_opmode_buffer);
            if(abs(v-0*pi/180)<0.1*pi/180)
```

```
                    break
                end;
            end;
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(4), vrep.
                simx_opmode_streaming);
            while true
                [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(4),
                    vrep.simx_opmode_buffer);
                if(abs(v-0*pi/180)<0.1*pi/180)
                    break
                end;
            end;
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(5), vrep.
                simx_opmode_streaming);
            while true
                [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(5),
                    vrep.simx_opmode_buffer);
                if(abs(v-0*pi/180)<0.1*pi/180)
                    break
                end;
            end;
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(6), vrep.
                simx_opmode_streaming);
            while true
                [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(6),
                    vrep.simx_opmode_buffer);
                if(abs(v-0*pi/180)<0.1*pi/180)
                    break
                end;
            end;
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(7), vrep.
                simx_opmode_streaming);
            while true
                [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(7),
                    vrep.simx_opmode_buffer);
                if(abs(v-0*pi/180)<0.1*pi/180)
                    break
                end;
            end;
            [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(8), vrep.
                simx_opmode_streaming);
            while true
                [err3,v]=vrep.simxGetJointPosition(clientID, jointHandles(8),
                    vrep.simx_opmode_buffer);
                if(abs(v-0*pi/180)<0.1*pi/180)
                    break
                end;
            end;
            vrep.simxFinish(clientID); %cerramos la conexion
else
            disp('Fallo al conectar con el servidor API remoto');
end;
vrep.delete(); %llamamos al destructor
```